FUNDAMENTOS DE INFORMÁTICA Y PROGRAMACIÓN CIENTÍFICA

Resolución en C y Matlab

Jesús María Zamarreño Cosme, María Teresa Alvarez Alvarez, Luis Felipe Acebes Arconada, Miguel Angel García Blanco, Fernando Juan Tadeo Rico

Agradecemos a la Consejería de Educación y Cultura de la Junta de Castilla y León el apoyo prestado a través de la ayuda para la elaboración de Recursos de Apoyo a la Enseñanza Universitaria, concedida a los autores.

1. Introducción

1.1 La importancia del ordenador en la ciencia

Hoy día, nuestra sociedad depende para su existencia, tal y como la conocemos, de una serie de desarrollos como la electricidad, el teléfono, etc. Nuestro estilo de vida se vería en gran medida afectado por la ausencia de alguno de estos "inventos". Sin embargo, no le damos excesiva importancia y asumimos que son tecnologías que "necesitamos" tener tanto para desarrollar nuestra actividad laboral, de ocio, etc. De la misma manera, existen hoy día multitud de actividades en las que la ausencia del ordenador sería calificado como un "desastre". De hecho, podemos poner como ejemplo de uno de estos "desastres" el denominado "Efecto 2000", problema que afectó a la inmensa mayoría de los ordenadores "veteranos" y cuya causa última se encuentra en un criterio desafortunado de representación y una falta de previsión por parte de los desarrolladores de software. Este "Efecto 2000" dio lugar a predicciones catastrofistas que auguraban un cataclismo en todos los ámbitos donde interviene el ordenador y que por suerte no se cumplieron.

Dejando de lado los efectos negativos que involucraría hoy día la ausencia de los ordenadores, fijémonos en los aspectos positivos y el papel tan destacado que ha tenido el ordenador en nuestra cultura y en particular en la ciencia.

Aunque estamos hablando de la importancia del ordenador (como máquina) en la actualidad, no debemos olvidar que un ordenador por sí mismo no es capaz de realizar una tarea útil, a no ser que sea correctamente instruido sobre las acciones que debe realizar. Es decir, realmente deberíamos hablar de la importancia de la **Informática**, entendiendo esta como una disciplina formada por el Hardware y el Software. Estos dos términos de difícil traducción al castellano se refieren a la parte física del ordenador (hardware) y a la parte lógica (software) que provoca que el ordenador realice determinadas tareas. Una vez clarificados estos conceptos, podemos declarar que la informática ha sido, y aún hoy día continúa siendo, el principal motor que impulsa el desarrollo de las ciencias y la tecnología.

1.2 Ámbitos científicos donde el ordenador es imprescindible

En general, los ordenadores son útiles en aquellas tareas en las que intervienen una serie de características, lo cual no implica que no puedan ser utilizados aún cuando dichas características no se encuentren presentes. Los ordenadores son especialmente útiles allí donde sea necesario tratar con:

- Grandes volúmenes de datos

- Datos comunes
- Repetitividad
- Distribución de la información
- Cálculos complejos
- Gran velocidad de cálculo

Comentemos brevemente algunos de estos puntos.

Existen tareas donde la **cantidad de datos** que es preciso procesar hace prácticamente inviable su tratamiento directamente por una o varias personas. Pensemos por ejemplo en la cantidad de información que se genera en los modernos aceleradores de partículas cuando tiene lugar una colisión. Los sistemas empleados para la captura de información deben ser muy rápidos y potentes para poder almacenar tantos datos como sea posible. Posteriormente es preciso realizar múltiples tratamientos y análisis sobre esos datos que se encontrarán almacenados en algún tipo de memoria masiva. Este tratamiento debe hacerlo un ordenador, ya que una persona dedicada a tiempo completo no sería capaz de acabar la tarea en cientos de años, aún suponiendo que dispusiera de calculadora. En este caso podríamos considerar a la calculadora como un ordenador altamente simplificado (en el siguiente capítulo daremos una definición precisa de ordenador y calculadora). Otro ejemplo del ámbito científico, donde el volumen de datos es enorme, lo podemos encontrar en el proyecto SETI@home[1] donde el enorme caudal de datos suministrado por los modernos telescopios es distribuido entre miles de ordenadores distribuidos por todo el mundo que los analizan y devuelven los resultados a un ordenador central.

En otros casos nos encontramos con que existen **datos comunes** que son utilizados por multitud de personas como pueden ser constantes de reacciones químicas, datos astronómicos, atmosféricos, etc. Ante esta situación es conveniente tener servidores centralizados que aglutinen estos datos, y que a medida que los investigadores de los diferentes campos necesiten acceder a ellos, recurran a estos servidores. De esta manera, se evita que la información esté duplicada y además se asegura que ésta sea coherente.

Un área donde los ordenadores son realmente útiles y que evitan un trabajo pesado al ser humano es en las **tareas repetitivas**. Se trata de tareas donde una misma acción se debe repetir miles o millones de veces. Pensemos por ejemplo en muchos de los métodos de optimización que se basan en un proceso iterativo el cual se debe repetir innumerables veces antes de llegar a la solución del algoritmo. La programación y su resolución por medio del

[1] Search for Extraterrestrial Intelligence at Home, búsqueda de inteligencia extraterrestre desde casa. http://setiathome.ssl.berkeley.edu

ordenador en estos casos reduce el tiempo de resolución cientos e incluso miles de veces. Además, una vez escrito el programa será posible aplicarlo a datos diferentes.

Se dice que estamos en la sociedad de la información, esto supone que se generan muchos datos, tanto científicos como de cualquier otro tipo. Para almacenar toda esta información sería preciso contar con un espacio de almacenamiento que pocas personas o empresas pueden costear. Teniendo en cuenta, además, que esta información cambia constantemente, su constante actualización requeriría amplios recursos. Para solucionar este problema y poder acceder a información actualizada y proveniente de múltiples fuentes, se recurre a redes de ordenadores interconectados donde un ordenador cualquiera de la red puede acceder a información almacenada en cualquier otro ordenador accesible a través de esa red. Es lo que denominamos **distribución de la información**.

Uno de los factores principales en el gran desarrollo de la ciencia tal y como la entendemos hoy día ha sido la capacidad de realizar **cálculos complejos** mediante los ordenadores. Muchas de las teorías físicas, químicas, etc. propuestas en décadas pasadas no ha sido posible corroborarlas hasta recientemente, debido a que entonces no existía potencia computacional para validarlas. En campos como la medicina, gracias a la capacidad de los ordenadores de realizar cálculos complejos, ha sido posible desarrollar nuevos métodos de diagnóstico como la tomografía computerizada. Podemos poner como ejemplo de la física el problema de las simulaciones multipartícula, donde para realizar una simulación realista de decenas de miles de partículas interactuando es preciso realizar del orden de 10^{14} cálculos en punto flotante.

Finalmente, un factor importante que hace a los ordenadores realmente útiles es la **velocidad** a la que son capaces de procesar los datos. La velocidad de proceso ha aumentado a medida que se desarrollaban nuevos microprocesadores; y la ciencia se ha podido enfrentar a nuevos retos como el control de procesos en tiempo real, la realidad virtual, las videoconferencias, etc.

En este apartado hemos visto las características que hacen al ordenador una herramienta fundamental en la ciencia moderna. Dentro de las diversas ciencias que existen, podríamos enumerar miles de aplicaciones en las que el ordenador se hace indispensable; como muestra enumeremos algunas:

- Ciencias físicas e ingeniería: resolución de ecuaciones diferenciales, integración numérica, simulación de sistemas, optimización, control, …
- Ciencias de la vida y médicas: diagnóstico médico, desarrollo de nuevos medicamentos, …

- Ciencias sociales y del comportamiento: evaluación de encuestas, análisis estadísticos de población, estudios de mercado, extrapolación de resultados, …
- Ingeniería con ayuda del computador: CAD (Computed Aided Design = Diseño asistido por ordenador), CAM (Computer Aided Manufacturing = Fabricación asistida por ordenador), DCS (Distributed Control System = Sistema de control distribuido), …

1.3 Enfoques teóricos, experimentales y computacionales

En cualquier rama de la ciencia, a la hora de resolver un problema determinado, existen y han existido tres enfoques fundamentales: enfoque teórico, experimental y computacional.

En el enfoque **teórico**, el investigador, usando el lenguaje de las matemáticas, desarrolla un modelo que explica y predice el comportamiento de la naturaleza. Un ejemplo de este enfoque podrían ser las leyes de Newton.

En el enfoque **experimental**, el investigador observa la naturaleza y adquiere datos medibles de su comportamiento. A partir de estos datos experimentales, trata de comprender su funcionamiento. Un ejemplo de este enfoque podría estar en los aceleradores de partículas, donde se experimenta con colisiones controladas y se analizan los datos resultantes para tratar de aprehender los elementos fundamentales de la materia.

A menudo los enfoques teórico y experimental se encuentran interrelacionados, en el sentido de que las teorías que se desarrollan es preciso corroborarlas con las observaciones, y los experimentos que se realicen deben servir para ajustar y validar las teorías.

Finalmente, en el enfoque **computacional**, el investigador recrea algún aspecto de la naturaleza en su ordenador utilizando conocimiento tanto teórico como práctico. Normalmente, se utiliza este enfoque cuando el enfoque experimental es costoso, difícil o imposible. El enfoque computacional puede dar lugar a nuevas teorías o sugerir nuevos experimentos. También sirve para validar teorías.

Un científico que haga uso del enfoque computacional debe dominar su disciplina, además de las matemáticas, el análisis numérico y la ciencia computacional. Debe ser hábil en el manejo de un sistema operativo que le permita ejecutar acciones sobre el ordenador y hábil en programación para expresar su problema en lenguaje computacional. Casi todos los lenguajes de programación se basan en unos principios comunes, por lo que el lenguaje que elija no es importante, aunque puede estar condicionado por su aprendizaje, la disponibilidad y el tipo de tarea que quiera llevar a cabo.

PARTE I

2. Funcionamiento del ordenador

2.1 Conceptos básicos y arquitectura

Este libro trata, como su título indica, de programación científica. Sin embargo, creemos que es importante tener unos conceptos básicos y claros del funcionamiento del ordenador para comprender ciertos aspectos importantes en programación como pueden ser algunas limitaciones de representación numérica, por ejemplo. Por lo tanto, en este capítulo nos centraremos en aquellos conocimientos básicos que toda persona que se "enfrenta" a un ordenador debe conocer, aún cuando intentaremos huir de conceptos particulares y que pueden quedar desfasados en pocos años.

2.1.1 Informática

En primer lugar, debemos preguntarnos por el significado y el campo de estudio del que se ocupa la **Informática**. En general, podemos decir que la Informática estudia diversos aspectos relacionados con la información, como son la adquisición, representación, tratamiento y transmisión, todo ello mediante ordenadores.

El origen del término Informática surge como unión de dos palabras:

> Informática ≡ INFORmación + autoMÁTICA

Según la R.A.E. (Real Academia Española), su definición oficial sería:

> Informática: "conjunto de conocimientos científicos y técnicos que hacen posible el tratamiento automático de la información por medio de ordenadores"

Es decir, que aparte de la necesidad del ordenador, es necesario tratar la información, entendida como cualquier conjunto de símbolos que represente hechos, objetos o ideas. Es decir, información es todo aquello que podemos representar con símbolos, siendo esta información clasificable atendiendo a múltiples factores y en distintos niveles. Así, podemos considerar como información los números, las palabras, las matrículas de coche, resultados de juegos olímpicos, características climatológicas de una región, etc.

2.1.2 Computador, computadora, ordenador

Como comentábamos en la definición anterior, la informática se encarga del tratamiento automático de la información, pero lo que es importante, mediante **ordenadores**. Los términos computador, computadora u ordenador se refieren a lo mismo, mas aunque se puedan emplear indistintamente, está más extendido el término ordenador y es el que utilizaremos en este libro.

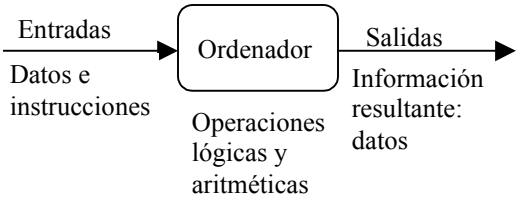


Figura 2-1. Papel del ordenador en el tratamiento de la información

Un **ordenador** es una máquina que es capaz de realizar operaciones complejas sobre la información sin intervención humana a partir de un programa de instrucciones. Vemos aquí una diferencia fundamental respecto a una calculadora, la cual también puede realizar operaciones complejas pero que necesita de una persona que la maneje. Un ordenador, sin embargo, trabaja ejecutando un programa de instrucciones previamente almacenado. De esta manera, podemos ver al ordenador como un elemento que recibe unas entradas, realiza unas operaciones, fundamentalmente lógicas y aritméticas, sobre ellas y devuelve unas salidas (Figura 2-1). Las entradas son simplemente datos, algunos de los cuales pueden ser instrucciones específicas para que opere el ordenador. La información que resulta de este procesamiento son nuevos datos que pueden volver a ser tratados o ser interpretados por el usuario final.

Por lo tanto, el ordenador trabaja con **datos** que aparecen representados por símbolos. Estos datos, como indicábamos antes cuando hablábamos de la *información*, pueden ser valores numéricos, hechos, objetos, ideas, etc. Lo importante es que puedan ser representados por símbolos para que el ordenador pueda procesarlos. Es habitual que los ordenadores trabajen con un subconjunto de todos los símbolos posibles, los denominados **caracteres**. Dentro del conjunto de caracteres, podemos distinguir tres subconjuntos:

- Numéricos: los dígitos del 0 al 9
- Alfabéticos: los caracteres alfabéticos de la *a* a la *z* tanto en mayúscula como en minúscula
- Especiales: aquí están incluidos todos los caracteres que no se incluyen en los subconjuntos anteriores, es decir, todo símbolo adicional que podamos necesitar a la hora de expresar una información: coma, punto y coma, dos puntos, signo de suma, resta, multiplicación, división, paréntesis, etc.

Resumiendo, podemos concluir que un ordenador trabaja con datos e instrucciones; las instrucciones le indican la tarea que debe realizar con los datos, obteniendo a su finalización nuevos datos, y los datos representan la información.

2.1.3 Codificación de la información

Un elemento clave en el funcionamiento del ordenador es lo que se llama **codificación de la información**.

> Codificación de la información: "Transformación que representa los elementos de un conjunto mediante los de otro, de forma tal que a cada elemento del primer conjunto le corresponde un elemento distinto del segundo"

Por lo tanto, la codificación de la información trata simplemente de una transformación de la información. ¿Por qué es importante en Informática? Es importante porque un ordenador solamente es capaz de almacenar y transferir la información en código binario, es decir, solamente maneja los valores 0 y 1, verdadero y falso, activado y desactivado. Sin embargo, las personas estamos acostumbradas a manejar otros códigos (caracteres para el texto, código decimal para las matemáticas, etc.). Por lo tanto, se hace necesaria la codificación de la información para poder traducir esta entre humanos y ordenadores.

La unidad elemental de información en Informática va a ser el BIT.

> BIT ≡ BInary digiT (dígito binario)

El bit se representa por la letra *b* minúscula, y lo podemos definir:

> bit: "posición o variable que toma el valor 0 ó 1"

Esta unidad de información es muy pequeña ya que con un único bit solamente podemos representar dos posibilidades: el circuito está abierto (1) o cerrado (0), el vaso está lleno (1) o vacío (0), verdadero (1) o falso (0), etc. Es habitual disponer de información que precisa de mayor "espacio de almacenamiento". En muchas ocasiones, en un ordenador querremos almacenar texto, ya que es la forma habitual en la que nos aparece la información. Por ello, se define un múltiplo del bit, con el que es posible almacenar un carácter, al que se denomina byte.

> byte: "número de bits necesarios para almacenar un carácter"

Actualmente, está bastante aceptado que un byte consta de 8 bits, por lo que también se denomina octeto. Esta unidad de información se representa con la *B* mayúscula para distinguirla del bit. Si calculamos todas las posibles combinaciones distintas que podemos formar con 8 bits, nos encontramos que son 2^8=256, empezando por los 8 bits a 0 (00000000) y terminando por todos a 1 (11111111), pasando por el resto de combinaciones de ceros y unos. Es decir, podemos representar 256 caracteres. Teniendo en cuenta que en el alfabeto castellano existen 27 caracteres distintos, tendríamos 54 entre minúsculas y mayúsculas, más los caracteres numéricos (0-9) tendríamos 66. Todavía nos quedarían todos los caracteres especiales, pero aun así puede parecer que 8 bits son demasiados para almacenar un carácter. Cuando veamos los códigos de E/S (Entrada/Salida) se verá la razón de utilizar 8 bits para almacenar un carácter, aunque existen códigos con solamente 7 bits. Sin embargo, esto no cambia para que haya permanecido como conversión fija:

1 byte (u octeto) = 8 bits

Nos podemos dar cuenta inmediatamente de que 1 byte es todavía una unidad de información muy pequeña: sólo nos permite representar un carácter. ¿Cuántos bytes necesitamos para almacenar una novela como El Quijote? Para cualquier texto que nos encontramos, el indicar su tamaño en bytes puede resultar pesado por ser en general números muy grandes. Por ello, es habitual emplear múltiplos del byte. Sin embargo, **debe tenerse cuidado** con estos múltiplos ya que su equivalencia es distinta que en otras ramas de la ciencia, como, por ejemplo, en la física. En física, el prefijo *kilo* equivale a *mil unidades*. En informática, *kilo* equivale a 2^{10} *unidades* que son 1024, algo más de mil. La razón de esta discrepancia se encuentra en que en el sistema decimal resulta muy útil trabajar con potencias de 10; sin embargo, en el sistema binario (que desarrollaremos más adelante), es más conveniente trabajar con potencias de 2, y la potencia de 2 más próxima a 1000 es 2^{10}. De la misma manera, para el prefijo *mega* que equivale a 10^6 en la física, en informática equivale a 2^{20} que son 1048576. Así tenemos:

- Kilo: 1 KB = 2^{10} bytes
- Mega: 1 MB = 2^{20} bytes
- Giga: 1 GB = 2^{30} bytes
- Tera: 1 TB = 2^{40} bytes
- Peta: 1 PB = 2^{50} bytes

2.1.4 Arquitectura Von Neumann

Aunque no nos vamos a detener en el análisis de un ordenador desde un punto de vista de arquitectura interna, sí es conveniente conocer al menos los elementos principales de que consta, así como algunas características que

hacen que un ordenador sea más potente que otro, aspecto fundamental a la hora de ejecutar cualquier algoritmo científico de mediana complejidad.

Existen multitud de arquitecturas a las que puede responder un ordenador en particular. En lugar de centrarnos en una arquitectura en particular (PC-XT, PC-AT, Macintosh, Sun, …), se suele considerar una arquitectura genérica y abstracta que explica bastante bien la forma de operar de un ordenador, aún cuando no exista en la realidad ningún ordenador que responda a esta tan simplista. Esta arquitectura se denomina arquitectura **Von Neumann** y aparece representada en la Figura 2-2.

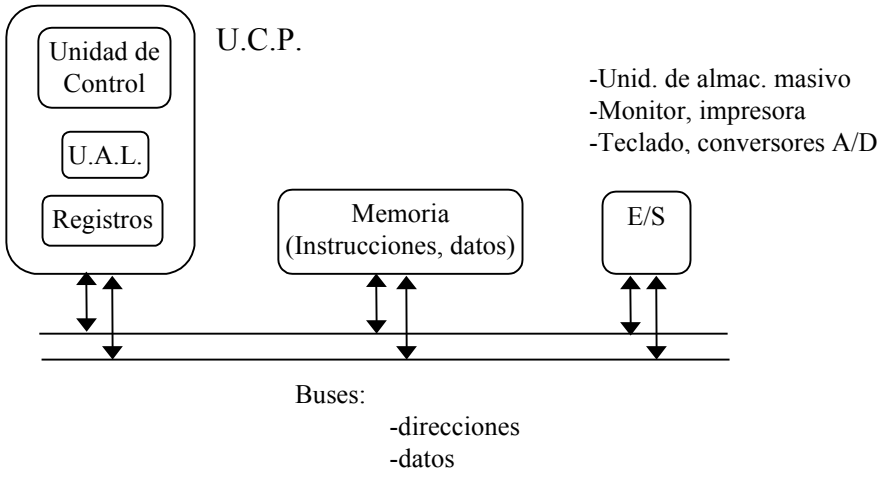


Figura 2-2. Arquitectura Von Neumann de un ordenador

La Unidad Central de Procesos (UCP) o **CPU** (Central Processing Unit) es el "cerebro" del ordenador y consta de tres partes básicas diferenciadas:

1. **Unidad de Control**: Se encarga de gestionar el secuenciamiento de las instrucciones, es decir, controlar el orden en que estas se ejecutan, activar señales en instantes adecuados para que se realice una tarea, etc.
2. **Unidad Aritmético-Lógica** (UAL) o **ALU** (Arithmetic-Logic Unit): Se encarga de realizar las operaciones de tipo aritmético y lógico.
3. **Registros**: Son zonas de almacenamiento temporal de datos. Los más importantes son:
    - PC (Program Counter, *Contador de Programa*): Contiene la dirección de memoria de la siguiente instrucción a ejecutar.
    - IR (Instruction Register, *Registro de Instrucción*): Donde se almacena la instrucción en ejecución.
    - Otros: Acumulador, Registro de Estado, etc.

La **memoria** del ordenador sirve para almacenar tanto datos como instrucciones. La memoria la podemos representar como una matriz con 2^m filas y n columnas. En cada celda se puede almacenar un 0 o un 1. Cada dirección (es decir, cada fila) da acceso a una posición o palabra de memoria de longitud *n*. Si *n=16*, en cada acción de escritura almacenaríamos una palabra de 16 bits, por lo que se precisa de 16 terminales en el dispositivo de memoria (los niveles 0 y 1 se representan eléctricamente en el ordenador como dos niveles distintos de tensión), más *m* terminales adicionales para indicar en cuál de las 2^m filas se almacena (con m combinaciones de ceros y unos, podemos representar 2^m elementos). Además, necesitamos un terminal, denominado R/W, para indicar si queremos leer (Read) o escribir (Write) en la memoria.

Cuando hablamos de la memoria del ordenador, habitualmente nos referimos a memoria RAM (Random Access Memory, memoria de acceso aleatorio) y ROM (Read Only Memory, memoria de solo lectura), aunque existen otras como la memoria caché. Cuando en un ordenador nos indican la memoria de que dispone, se suele referir a la memoria RAM, que es en la que almacenamos los programas en ejecución y los datos en uso.

En la Figura 2-2 aparece también un bloque denominado E/S que se refiere a los dispositivos de Entrada/Salida. En este bloque aparecen todos los elementos distintos de la CPU y la memoria, y que permiten al ordenador comunicarse con el exterior, bien recibiendo datos a través de teclados, ratones, tabletas digitalizadoras, sensores de temperatura, etc., bien enviando datos a otros como monitores, altavoces, impresoras, etc. Existen también unos dispositivos que se pueden considerar tanto como de E/S como de memoria masiva y que sirven para almacenar y acceder a grandes cantidades de información como son los discos duros, discos ópticos, cintas magnéticas, etc.

Los distintos elementos del ordenador se comunican a través de los denominados **buses**. Estos son simplemente *pistas eléctricas* a través de las cuales se envían datos unos dispositivos a otros.

2.1.5 Otras definiciones

Finalizamos esta primera parte sobre el funcionamiento del ordenador dando unas definiciones y conceptos de uso habitual.

> Periféricos: "conjunto de unidades de E/S y memoria masiva (dispositivos de almacenamiento como discos duros)"

La unidad de control (dentro de la CPU) contiene un reloj interno o generador de pulsos que le permite sincronizar los distintos elementos. La frecuencia de

este reloj suele ser del orden de MHz y es una medida de la rapidez con que es capaz de operar la CPU.

> Velocidad de transmisión: "cantidad de información transferida por segundo entre una unidad y otra"

La velocidad de transmisión es también un indicador de la velocidad del ordenador, ya que aunque la CPU sea muy rápida, si los datos no llegan a suficiente velocidad, no sirve de nada. La velocidad de transmisión se suele indicar en MB/s.

> Longitud de palabra: "número de bits transmitidos simultáneamente"

La longitud de palabra también es importante a la hora de estimar la capacidad computacional de un ordenador ya que si somos capaces de transmitir 64 bits simultáneamente en lugar de 32, habremos doblado la velocidad de transmisión de los datos.

> Microcomputador: "ordenador cuyo procesador central (CPU) es un microprocesador"
>
> Microprocesador: "uno o varios circuitos integrados que realizan las funciones de un procesador central"

Daremos finalmente unas definiciones básicas que hacen referencia más a la parte software que a la parte hardware, aunque evidentemente, para que el ordenador ejecute una orden que le demos por software, deberá suceder algo a nivel de hardware para que se lleve a cabo.

> Instrucción: "conjunto de símbolos que representa una orden de operación o tratamiento para el ordenador"

Las instrucciones pueden ser de varios tipos:

- De transferencias de datos: instrucción que provoca que un dato simplemente fluya de una posición a otra, por ejemplo de la memoria a un registro, de una posición de memoria a otra, ...
- De tratamiento: instrucción que realiza alguna operación sobre un/os dato/s, por ejemplo, una suma, una resta, un incremento, ...
- De control de flujo o de bifurcación y saltos: instrucción que provoca que una ejecución secuencial de instrucciones se interrumpa y se ejecute una instrucción que no es la siguiente en el flujo habitual de un programa (definiremos programa a continuación), por ejemplo, dar un error cuando un número del cual se va a calcular la raíz cuadrada sea negativo.

- Otras: instrucciones que no encajan en alguno de los tipos anteriores como puede ser la instrucción nula (no hace nada) o la de finalización del programa.

> Programa: "conjunto ordenado de sentencias que se dan al ordenador indicándole las operaciones o tareas que se desea realice"

Un programa consta de sentencias. Las sentencias pueden ser de dos tipos: imperativas o declarativas. Las sentencias imperativas son las instrucciones ya que obligan al ordenador a realizar una acción. Las sentencias declarativas simplemente indican al ordenador (realmente, al compilador, que veremos a continuación) el tipo de variables y funciones que va a usar.

> Lenguaje de programación: "símbolos y reglas para construir programas"

Un lenguaje de programación, al igual que un lenguaje natural como el inglés, francés o castellano, nos indica los símbolos que podemos utilizar y las reglas que debemos seguir para entendernos. Por ejemplo, para realizar una asignación a una variable, el lenguaje de programación nos indicará que debemos realizarlo como *a=4*, o *a:=4*, o *a=4;*.

Ya hemos explicado que el ordenador sólo es capaz de operar con ceros y unos. El lenguaje que utiliza el ordenador, basado en ceros y unos, para realizar las tareas, se denomina **lenguaje máquina**. Una instrucción en lenguaje máquina suele constar de un código de operación y un campo de dirección. El código de operación indica de qué instrucción se trata y en el campo de dirección habitualmente se encuentran los posibles operandos sobre los que actúa la instrucción.

Como sería muy tedioso realizar un programa con instrucciones del tipo 0010010010000011 (aunque hay gente que lo realiza a partir de lenguaje ensamblador), se desarrollaron los **lenguajes de alto nivel** como BASIC, FORTRAN, C, PASCAL, JAVA, C++, etc. Estos lenguajes están más próximos al lenguaje natural (aunque son más restrictivos y estrictos) por lo que facilitan la labor de desarrollar un programa. Evidentemente, el lenguaje de alto nivel no lo entiende directamente el ordenador, por lo que se hace necesario un intermediario, un **traductor** que transforme el programa en lenguaje de alto nivel al lenguaje máquina para que el ordenador lo pueda ejecutar. Existen dos tipos de traductores:

- *Compiladores*: Realizan la traducción del programa completo, generando el programa en código máquina como ente independiente. Se podría comparar con un traductor al que se le da una página en inglés y nos

devuelve su traducción al castellano en otra página al cabo de un rato. Un ejemplo de lenguaje de este tipo es C.

- *Intérpretes*: Van ejecutando el programa mediante la traducción instrucción a instrucción del programa original. Se podría comparar con una traducción simultánea. Un ejemplo de lenguaje de este tipo es BASIC.

Finalmente, debemos hacer mención del sistema operativo, elemento clave para poder manejar un ordenador. Un ordenador consta de elementos electrónicos que se activan mediante pulsos eléctricos en secuencias apropiadas. Sería absurdo que para escribir un dato en un disco duro tuviéramos que dar las órdenes de girar al disco, posicionar la cabeza magnética, controlar el punto en el que grabamos el dato, etc. Para facilitar el uso del ordenador por parte de las personas existe el sistema operativo definido como:

> Sistema operativo: "conjunto de programas que controlan y gestionan los recursos del ordenador"

Por lo tanto, el sistema operativo es un elemento software más que se está ejecutando continuamente y que nos permite interactuar con el ordenador de forma "amigable". La forma de interactuar con el ordenador se realiza mediante un **lenguaje de control** propio del sistema operativo que consta de órdenes o comandos. De esta manera, para copiar un fichero de una ubicación a otra usaríamos la orden del sistema operativo *COPY* o *cp* según se trate de sistema operativo MS-DOS o UNIX, respectivamente. Desarrollaremos con más detalle el tema de los sistemas operativos en el tema siguiente.

2.2 Representación de la información: sistemas de numeración, operaciones y representación interna

Como se ha visto en el apartado anterior, un ordenador se encarga de ejecutar programas, estando estos compuestos por instrucciones (qué debe hacer) y datos (sobre qué actuar).

> Programa = Instrucciones + Datos

A la hora de representar esta información en el ordenador, debemos establecer un conjunto de símbolos común para que podamos entendernos con el ordenador, es decir, para que sepamos qué significado tiene la secuencia de ceros y unos que está almacenada en la memoria del ordenador. Desde un punto de vista del usuario, es deseable manejar la información a través de caracteres, pudiendo clasificarlos en los siguientes tipos:

- *Alfanuméricos*. Incluyen los dos tipos siguientes:
  - ✓ Alfabéticos. Letras del alfabeto tanto en minúscula como en mayúscula

- ✓ Numéricos. Dígitos

- *Especiales*. Caracteres adicionales (coma, paréntesis, asterisco, …)
- *De control*. Caracteres no representables pero que tienen significado en un texto como tabulaciones, retornos de carro, nueva línea, fin de fichero, etc.
- *Gráficos*. Caracteres que representan algún gráfico o parte de un gráfico (líneas verticales, horizontales, triángulos, círculos, etc.)

Sin embargo, desde un punto de vista interno del ordenador, su representación está basada en secuencias de ceros y unos. Por lo tanto, para que máquina y persona puedan entenderse se necesita una correspondencia entre los conjuntos:

$\alpha \equiv \{A,B,\ldots,Z,a,b,\ldots,z,0,1,2,\ldots 9,/,+,(,\ldots\}$

y

$\beta \equiv \{0,1\}^n$

de manera que a cada elemento del primer conjunto le corresponda uno del segundo conjunto, y así poder hacer la traducción. El conjunto β está compuesto por todas las combinaciones posibles de ceros y unos de longitud total *n*. Es decir, si *n=3*, el conjunto β sería {000,001,010,011,100,101,110,111}, con lo que sería posible realizar una correspondencia de solamente $2^3=8$ caracteres.

A la hora de establecer esta correspondencia, dependiendo de cómo la establezcamos tendremos un código u otro. Además, podemos distinguir dos tipos de código en función de su orientación:

- **Códigos de E/S**. Establecen una correspondencia entre los caracteres y el conjunto β. Un ejemplo de código de E/S es el famoso código ASCII que hace corresponder, por ejemplo, a la letra A con el valor 1000001.
- **Códigos binarios**. Estos códigos están orientados a la representación de valores numéricos. Es decir, el número 15 lo podríamos ver como una cadena de texto en cuyo caso podríamos hallar su código ASCII (sería 0110001 0110101) o como un número entero sin signo en cuyo caso su código sería 1111, o incluso como un número real en cuyo caso su representación sería distinta.

2.2.1 Sistemas de numeración y operaciones

Vamos a comenzar viendo los sistemas de numeración más utilizados en Informática y las razones de su uso. También veremos cómo realizar operaciones aritméticas en estos sistemas de numeración. Veremos el sistema de numeración por excelencia en Informática: sistema de numeración en base dos o binario o binario natural. También veremos los denominados códigos

intermedios: octal y hexadecimal, que permiten abreviar de forma cómoda la representación binaria.

Antes de introducirnos en estos sistemas de numeración, vamos a analizar el problema de representación de valores numéricos, viendo en particular el caso concreto del sistema de numeración decimal, que es el que estamos acostumbrados a manejar desde la infancia.

Representación

Cuando hablamos de un **sistema de numeración de base *b*** nos referimos a que éste está formado por un alfabeto compuesto de *b* símbolos. Cuando formamos un número con este alfabeto (considérese el número decimal 232), cada cifra contribuye al valor total del número dependiendo de dos factores:

- la cifra en sí
- la posición dentro del número

De esta manera, en el ejemplo que planteábamos (232), la contribución al número total no es la misma la del primer 2 que la del situado en última posición, el primero contribuye con un valor 200 al valor total y el último con sólo 2. Es decir, es importante la posición de la cifra dentro del número. Evidentemente, también es importante la cifra en sí, es decir, no es lo mismo que la primera cifra sea un 2 a que sea un 5.

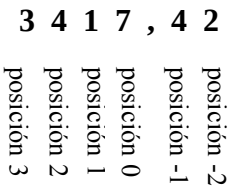
Analicemos un poco más el caso del sistema de numeración decimal. Diremos que la base es 10, *b=10* ya que el alfabeto que podemos usar para formar los números está compuesto de 10 símbolos:

Alfabeto (*b=10*) : {0,1,2,3,4,5,6,7,8,9}

Cuando formamos un número en este sistema de numeración podemos analizar la contribución de cada cifra en función de la posición de la siguiente manera (ejemplo para el número 3417,42):

$3417{,}42 = 3\cdot10^{3}+4\cdot10^{2}+1\cdot10^{1}+7\cdot10^{0}+4\cdot10^{-1}+2\cdot10^{-2}$

Es decir, cada posición del número "pesa" de manera distinta. Estas posiciones se numeran desde la coma decimal hacia la izquierda comenzando por cero. De la coma decimal hacia la derecha, las posiciones se numeran con valores negativos. Es decir:



En el sistema de numeración decimal, las diversas posiciones del número tienen nombres particulares, por ejemplo, la posición 0 se denomina posición de las unidades y contribuye con $b^0=10^0$, la posición 1 se denomina posición de las decenas y contribuye con $b^1=10^1$, y así sucesivamente.

Para un sistema de numeración genérico de base *b*, la forma de obtener el valor de un número se realiza de forma semejante, la única diferencia será el alfabeto que utilizamos y los diversos pesos que no serán potencias de 10, sino potencias de *b*.

De esta manera, un número N en cualquier sistema de numeración lo veríamos como:

$N \equiv \ldots n_4\ n_3\ n_2\ n_1\ n_0\ ,\ n_{-1}\ n_{-2}\ \ldots$

y su valor lo podemos calcular como:

$$\text{Valor} = \ldots + n_4 \cdot b^4 + n_3 \cdot b^3 + n_2 \cdot b^2 + n_1 \cdot b^1 + n_0 \cdot b^0 + n_{-1} \cdot b^{-1} + n_{-2} \cdot b^{-2} + \ldots$$

Es importante señalar que de esta manera obtenemos el valor del número (aunque sea una propiedad del número, no del sistema de numeración que utilicemos, es decir, 5 puntos seguirán siendo 5 independientemente de la forma en que representemos dicho valor) en el sistema decimal. Es decir, el método propuesto para calcular el valor de un número nos permite convertir un número dado en un sistema de numeración cualquiera al sistema de numeración decimal.

También debemos señalar que a la hora de indicar en qué sistema de numeración está dado un número, la forma de indicarlo es terminar el número con el símbolo de 'cerrar paréntesis' y un subíndice con la base del sistema de numeración *b*. Por ejemplo, en el caso anterior, el número lo representaríamos como $3417{,}42)_{10}$. Esto es importante ya que si no lo indicáramos, dicho número podríamos pensar que podría estar dado en sistema de numeración octal (que veremos más adelante). En ciertas ocasiones el sistema de numeración está implícito y no hará falta ponerlo, pero para evitar confusiones conviene indicarlo, sobre todo en Informática.

Ejemplo

Sea el sistema de numeración de base 5 formado por el alfabeto {0,1,2,3,4}. Calcular el valor del número $34{,}4)_5$.

Como la base es b=5, sumamos el valor de cada cifra multiplicado por el peso de su posición,

Valor de $34,4)_5 = 3 \cdot 5^1 + 4 \cdot 5^0 + 4 \cdot 5^{-1} = 15 + 4 + 0,8 = 19,8$

Este valor que obtenemos es la conversión del número a base 10, es decir,

$34,4)_5 = 19,8)_{10}$ o

En este momento nos podríamos preguntar cómo podemos, dado el valor de un número, hallar su representación en un sistema de numeración de base *b* cualquiera. O lo que es lo mismo, dado un número en sistema de numeración decimal, realizar la conversión a base *b*. Si nos fijamos en la forma de calcular el valor de un número, que es su representación en base 10, y nos fijamos en la parte entera del número:

$$\text{Valor} = \ldots + n_4 \cdot b^4 + n_3 \cdot b^3 + n_2 \cdot b^2 + n_1 \cdot b^1 + n_0 \cdot b^0$$

nos damos cuenta que dividiendo esta cantidad por la base *b* resultaría el cociente $\ldots + n_4 \cdot b^3 + n_3 \cdot b^2 + n_2 \cdot b^1 + n_1 \cdot b^0$ quedando un resto de n_0, es decir, el primer resto que nos queda es el coeficiente n_0 del número representado en base *b*. Si la cantidad que nos ha quedado la volvemos a dividir por *b* podemos comprobar que nos vuelve a quedar un resto, en este caso igual a n_1, es decir, el siguiente coeficiente. Esto lo podemos repetir dividiendo sucesivamente por *b* hasta llegar al último valor menor que *b* que corresponderá con el coeficiente de mayor peso del número. En cuanto a la parte fraccionaria:

$$n_{-1} \cdot b^{-1} + n_{-2} \cdot b^{-2} + \ldots$$

el procedimiento para, dado el valor resultante (su representación en base 10), obtener los diversos coeficientes, consiste en multiplicar la parte fraccionaria por la base *b*. Efectivamente, al multiplicar la anterior cantidad por *b*, nos queda $n_{-1} \cdot b^0 + n_{-2} \cdot b^{-1} + \ldots$ con lo que la parte entera de dicho número representa el coeficiente n_{-1}. A continuación nos quedaríamos de nuevo con la parte fraccionaria resultante ($n_{-2} \cdot b^{-1} + n_{-3} \cdot b^{-2} + \ldots$) que volveríamos a multiplicar por *b* para que quede como parte entera n_{-2}, y así sucesivamente hasta que la parte fraccionaria sea exactamente 0 (en cuyo caso la conversión es exacta) o hasta que el número de coeficientes se considere suficiente.

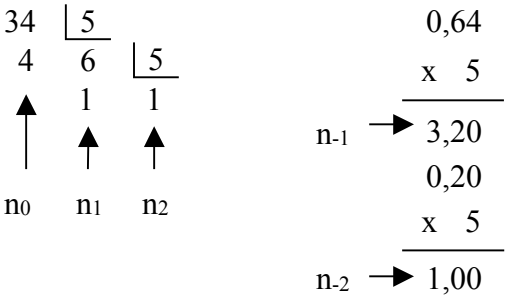
Para clarificar el procedimiento, expongamos el siguiente ejemplo.

Ejemplo

Sea el sistema de numeración de base 5 formado por el alfabeto {0,1,2,3,4}. Calcular la representación en este sistema de numeración del valor decimal $34,64)_{10}$.

El procedimiento visto anteriormente para convertir un número expresado en base 10 a otro sistema de numeración nos indica que la parte entera y la fraccionaria deben tratarse por separado. Dividimos 34 por la base (5) y nos fijamos en los restos (el número lo formamos tomándolos de derecha a

izquierda comenzando por el último cociente). Multiplicamos 0,64 por la base (5) y en cada ocasión nos quedamos con la parte entera y repetimos la operación con la parte fraccionaria (el número lo formamos tomando por orden las partes enteras):



En este caso, al hacer la conversión, nos resulta un número finito de coeficientes para la parte fraccionaria ya que hemos llegado a que esta es 0 por lo que no tenemos que extraer más coeficientes. El resultado de la conversión es:

$34{,}64)_{10} = 114{,}31)_5$ o

En la conversión que hemos visto en el ejemplo anterior son importantes varios detalles que a menudo son causa de errores y que no se deben olvidar:

Parte entera:

- El último cociente TAMBIÉN se considera
- Los coeficientes se extraen de derecha a izquierda (primero n_2, después n_1 y finalmente n_0)

Parte fraccionaria:

- Al extraer los valores enteros, continuar solamente con la parte fraccionaria (de 3,20 extraemos el 3 y continuamos con 0,20)
- Si al multiplicar por la base nos queda 0,xx extraemos un 0 y continuamos con 0,xx

En los subapartados siguientes, veremos los sistemas de numeración más utilizados en Informática.

Base dos

Una vez visto el caso general de un sistema de numeración de base *b*, no haría falta considerar los casos particulares, mas por claridad y por su amplio uso en Informática, vamos a comenzar por la base dos. En este caso, la base es

$$b = 2$$

por lo que el alfabeto consta de dos elementos:

$$\{0,1\}$$

El sistema de numeración de base dos también es conocido como sistema de numeración **binario** o **binario natural**.

Para pasar de **binario a decimal**, aplicamos la regla conocida, teniendo en cuenta que ahora nos encontramos con pesos potencia de 2 (la base).

Ejemplo

Obtener el valor en decimal del número binario $110100,01)_2$.

$$110100,01)_2 = 1\cdot2^5+1\cdot2^4+0\cdot2^3+1\cdot2^2+0\cdot2^1+0\cdot2^0+0\cdot2^{-1}+1\cdot2^{-2} =$$

$$1\cdot2^5+1\cdot2^4+1\cdot2^2+1\cdot2^{-2} = 52,25)_{10}$$ o

También es común tener en mente las diversas potencias de dos y sumar las de aquellas posiciones donde aparezca un 1.

Ejemplo

Obtener el valor en decimal del número binario $1001,001)_2$.

$$1\ 0\ 0\ 1\ ,\ 0\ 1\ )_2$$

$$8\ 4\ 2\ 1\ ,\ ½\ ¼\ )_2$$

$$= 8 + 1 + ¼ = 9,25)_{10}$$

o

Para pasar de **decimal a binario**, aplicamos el procedimiento descrito para pasar de decimal a sistema en base *b* particularizándolo a *b=2*.

Ejemplo

Obtener el valor en binario del número decimal $26,1875)_{10}$.

Convertimos la parte entera:

26	| 2			
0	13	| 2		
	1	6	| 2	
		0	3	| 2
			1	1

que resulta ser 26)$_{10}$ = 11010)$_2$. A continuación convertimos la parte fraccionaria:

0,1875	0,3750	0,7500	0,5000
x 2	x 2	x 2	x 2
0,3750	0,7500	1,5000	1,0000

que resulta ser 0,1875)$_{10}$ = 0,0011)$_2$. La conversión del número completo es la concatenación de ambas partes: entera y fraccionaria.

26,1875)$_{10}$ = 11010,0011)$_2$ o

A la hora de *contar* en binario, se realiza de la misma manera que en cualquier sistema de numeración, es decir, se va aumentando el coeficiente de la posición 0, a continuación se aumenta el de posición 1 y, con éste fijo, se aumenta el de posición 0, y así sucesivamente. En decimal, cuando llegamos al 09, aumentamos el coeficiente de posición 1 e iniciamos el *conteo* de la posición 0, es decir, pasamos al 10. En binario es igual, pero hay que tener en cuenta que sólo tenemos dos elementos para contar, el 0 y el 1. De cualquier manera, a la hora de plantear una tabla de equivalencia entre decimal y binario, bien la obtenemos por conversión de la manera descrita anteriormente, o por conteo simultáneo en ambos sistemas de numeración. Si nos limitamos a 3 bits, la **tabla de equivalencia decimal-binario** quedaría:

Decimal	Binario
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

Operaciones aritméticas en base 2

Las operaciones aritméticas se efectúan de la misma manera en cualquier sistema de numeración ya que el resultado debe ser el mismo independientemente del sistema de numeración utilizado. La principal

dificultad al plantear operaciones aritméticas en base 2 estriba en la novedad. Lo difícil es darse cuenta de cómo las realizamos en sistema decimal, debido a que después de tantos años operando en decimal, realizamos las operaciones de forma automática sin pararnos a pensar en cómo las hacemos. Cuando sumamos dos cifras en una suma de dos números y el resultado supera la representación en una sola cifra, decimos que nos llevamos 1 que lo consideramos en la posición siguiente. Cuando restamos dos cifras en las que el minuendo es menor que el sustraendo, realizamos la operación considerando 10+minuendo-sustraendo y decimos que nos llevamos 1 que lo consideramos en la posición siguiente. La multiplicación y la división tienen menos dificultades, siendo estas fruto exclusivamente de la representación binaria a la que no estamos acostumbrados. En base 2, las posibilidades que nos podemos encontrar al realizar una suma, resta, multiplicación o división entre dos números son pocas, en realidad 4, que son todas las posibles combinaciones que tenemos. Es por ello, que podemos plantear unas tablas que nos ayudarán a realizar estas operaciones.

Suma aritmética

a	b	a + b
0	0	0
0	1	1
1	0	1
1	1	0 y me llevo 1

Resta aritmética

a	b	a - b
0	0	0
0	1	1 y me llevo 1
1	0	1
1	1	0

La suma de 0+0, 0+1 y 1+0 no plantea ninguna dificultad. Cuando nos encontramos con 1+1, el resultado será 10 por lo que el resultado de la operación en dicha cifra dentro de una suma es 0 y debemos considerar el 1 en la siguiente posición. Al restar, no plantea ningún problema las operaciones 0-0, 1-0 y 1-1. Cuando restamos 0-1, debemos hacer la operación 10-01 que resulta ser 1 aunque el 1 que hemos añadido al minuendo lo debemos restar en la siguiente posición.

La multiplicación aritmética no plantea ningún tipo de dificultad al coincidir con lo que ya sabemos en decimal. Lo mismo podemos decir de la división, aunque hay que tener en cuenta que cuando hagamos una división de dos números en binario, nos aparecerán restas que debemos resolver.

Con el objetivo de clarificar un poco más la forma de realizar operaciones aritméticas con números en binario de cualquier longitud, vamos a poner un ejemplo de cada una de las operaciones en las que haremos uso de las tablas previamente expuestas.

Multiplicación aritmética		
a	b	a · b
0	0	0
0	1	0
1	0	0
1	1	1

División aritmética		
a	b	a / b
0	0	indeterminación
0	1	0
1	0	∞
1	1	1

Ejemplo

Realizar la suma de los números binarios 1110101 + 1110110.

```
   1 1 1 0 1 0 1
+  1 1 1 0 1 1 0
 ---------------
 1 1 1 0 1 0 1 1
```

Es de destacar que en la posición 2 hemos sumado 1+1=0 y nos llevamos 1 que sumamos en la siguiente posición: 0+0+1=1. En la quinta posición nos aparece un 1 que arrastramos de la posición 4, por lo que tenemos 1+1+1 = (0 y me llevo 1)+1 = 0+1 y me llevo 1 = 1 y me llevo 1. o

Ejemplo

Realizar la resta en binario 1101010 – 1010111.

```
   1 1 0 1 0 1 0
-  1 0 1 0 1 1 1
 ---------------
   0 0 1 0 0 1 1
```

En este caso es de destacar que fruto de la operación en la posición 0, en la que realizamos 0-1 = 1 y me llevo 1, en la posición 1 debemos hacer 1-1 = 0 pero como nos llevamos 1, al resultado de esa posición le debemos restar 1, es decir, (1-1)-1 = 0-1 = 1 y me llevo 1. En la posición 2 tenemos que tener en cuenta que nos llevamos también 1, por lo que la operación es (0-1)-1 = (1 y

me llevo 1)-1 = (1-1) y me llevo 1 = 0 y me llevo 1. El resto es inmediato teniendo en cuenta estas explicaciones. o

---

Ejemplo

Realizar la multiplicación en binario 1010011 x 10.

```
      1 0 1 0 0 1 1
            x 1 0
  ------------------
      0 0 0 0 0 0 0
+   1 0 1 0 0 1 1
  ------------------
    1 0 1 0 0 1 1 0
```

o

Ejemplo

Realizar la división en binario 1101,01 / 101.

```
   1101,01 |101
 - 101     -----
  ---      10,101 .....
  0011 0
  - 10 1
  -----
   00 110
    - 101
    -----
      001
        \
         \
```

o

Como se ha podido comprobar, la multiplicación se realiza exactamente de la misma manera que en decimal, con la salvedad de realizar las diversas sumas en binario. De la misma manera, la división se realiza de forma análoga teniendo en cuenta la forma de realizar las restas en binario.

Representación en complementos

Uno de los conceptos más interesantes dentro de la representación de la información en el ordenador es el concepto de **complemento**. Su importancia será evidente al final del capítulo. En este punto podemos decir que su utilidad

vendrá dada al capacitarnos convertir las restas en sumas. ¿Por qué queremos hacer eso?

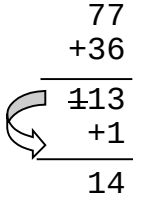
En primer lugar, está el tema de la representación: hemos dicho que el ordenador almacena y trata la información como secuencias de ceros y unos, así que ¿cómo representamos un número negativo? En "papel" simplemente ponemos un signo menos delante del número, pero si sólo disponemos de ceros y unos… Este tema de la representación lo veremos más adelante cuando hablemos de representación interna de la información (apartado 2.2.4).

En segundo lugar, si pudiéramos convertir de manera sencilla las restas en sumas, la unidad aritmético lógica solamente necesitaría "saber" sumar, lo cual implicaría un diseño más sencillo ya que como hemos visto es mucho más fácil sumar que restar.

Comencemos por dar la definición de complemento a la base menos uno (el concepto de complemento sirve para cualquier sistema de numeración, no nos limitamos al sistema binario).

> Complemento a la base menos uno de un número N: "es el número que resulta de restar cada una de las cifras de N a la base menos uno del sistema de numeración que se esté utilizando"

Para facilitar la comprensión, comenzaremos por aplicar esta definición en base 10. En este caso, nos planteamos el cálculo del complemento a 9 de un número cualquiera, por ejemplo, el 63. Tenemos que restar cada cifra del número a la base menos uno, es decir, 99 – 63 que son 36. Por lo tanto, el complemento a 9 de 63 es 36. Hemos indicado anteriormente que una de las utilidades es convertir restas en sumas. Supongamos que queremos restar 77 – 63. Es fácil realizar la operación y concluir que son 14. Esta misma operación se puede realizar como una suma sin más que sustituir el 63 por su complemento a 9 y sumarlos, teniendo en cuenta que si hay acarreo[2], este se debe sumar al resultado. Es decir, la operación que debemos hacer es:



[2] se entiende por acarreo la cifra situada en la posición n+1 al realizar una operación en la cual intervienen números de *n* cifras.

que resulta ser igual a 77 – 63, como esperábamos.

Esta propiedad que hemos visto de los complementos a la base menos es común a todos los sistemas de numeración. Nosotros estamos interesados en base 2. El cálculo de complementos en esta base es realmente sencillo. Si, por ejemplo, queremos calcular el complemento a uno de 10010, solamente tenemos que restar todas las cifras de la base menos 1, es decir, debemos realizar la operación 11111 – 10010 = 01101. El complemento a uno de 10010 es 01101. Se supone que esta operación de complemento la estamos realizando asumiendo un entorno de trabajo de 5 bits. Es fácil comprobar que la operación sería distinta si consideráramos que trabajamos con 6 bits, en cuyo caso deberíamos realizar 111111 – 010010, que como se puede comprobar es distinto. Por lo tanto, es **importante** tener en cuenta el número de bits con el que trabajamos al realizar el complemento.

Por otra parte, fijémonos en el resultado que hemos obtenido: el complemento a uno de 10010 es 01101. A primera vista, se puede comprobar que estos dos números guardan una relación sencilla: el complemento a uno de un número binario se puede obtener cambiando los ceros por unos y los unos por ceros, es decir, como veremos más adelante, consiste en realizar la operación lógica NOT. Por lo tanto, una resta en binario se realiza de forma más sencilla sumando el minuendo al complemento a uno del sustraendo y si existe acarreo se suma al resultado.

Ejemplo

Realizar la resta 1000111 – 10010 haciendo uso del complemento a uno del sustraendo.

Como tenemos que trabajar con 7 bits, debido al tamaño del minuendo, debemos realizar el complemento a uno de 0010010 que resulta ser 1101101, por lo que ya podemos realizar la operación, donde comprobaremos que realizando la resta directamente, obtenemos el mismo resultado.

```
  1000111          1000111
- 0010010        + 1101101
---------        ---------
  0110101        ~10110100
                       + 1
                 ---------
                  0110101
```

o

Hemos visto la definición y utilidad del complemento a la base menos uno. También existe un elemento relacionado con este, denominado **complemento a la base**.

> *Complemento a la base de un número N: "es el número que resulta de restar cada una de las cifras del número N a la base menos uno del sistema que se esté utilizando y, posteriormente, sumar uno a la diferencia obtenida"*

Es decir, en base 2, hablaríamos de complemento a 2. La utilidad de este complemento es la misma que el complemento a 1. Aparte de servirnos para representar números negativos como veremos en la representación interna de la información, nos permite realizar las restas como si fueran sumas de manera análoga a la vista anteriormente. El complemento a 2 de 0010010 es (1101101+1) = 1101110 para el ejemplo anterior; en el que para obtener el mismo resultado debemos sumar el minuendo al complemento a 2 del sustraendo, y, si existe acarreo, suprimirlo. ¡Compruébese!

Operaciones lógicas

Acabamos de ver las operaciones aritméticas típicas realizadas sobre un sistema de numeración en base 2. En este sistema de numeración es posible plantear también otro tipo de operaciones denominadas operaciones lógicas. En el apartado 2.3 veremos otros conceptos de lógica binaria. En este punto, simplemente veremos las operaciones típicas.

En el sistema binario, disponemos de solamente dos elementos, cero y uno. Con un bit (unidad mínima de información) podemos hacer referencia a variables que solamente toman dos valores posibles: encendido / apagado, abierto / cerrado, verdadero / falso. De hecho, es habitual trabajar en este sistema de numeración con los valores V (verdadero = 1), F (falso = 0). Debido a esto, se pueden hacer razonamientos verdadero / falso a partir de premisas que pueden ser verdaderas o falsas, es decir, estamos utilizando la **lógica**.

Las operaciones lógicas, aun cuando se apliquen a números de varios bits, se realizan bit a bit, entre bits situados en la misma posición. Es decir, no sucederá como en las operaciones aritméticas, en las que nos "llevábamos una".

Existen 16 operaciones lógicas posibles entre dos variables lógicas[3], aunque en este punto nos centraremos en las más conocidas y utilizadas.

Para ver la forma de operar de cada operador lógico, plantearemos la *tabla de operación* o *tabla de verdad*. Esta tabla es análoga a las que vimos para las

[3] Variable lógica: variable que puede tomar los valores verdadero o falso.

operaciones aritméticas: se plantean todas las posibilidades entre operandos y para cada combinación se muestra el resultado del operador.

La operación lógica OR da como resultado 1 si y sólo si alguno de los dos operandos es 1 (o ambos). Cuando utilizamos este operador, lo que estamos diciendo es que el resultado será verdadero cuando al menos uno de los operandos sea verdadero. Compárese con la suma aritmética para ver por qué esta también se denomina *suma*.

Suma lógica: OR

a	b	a OR b
0	0	0
0	1	1
1	0	1
1	1	1

Producto lógico: AND

a	b	a AND b
0	0	0
0	1	0
1	0	0
1	1	1

La operación lógica AND da como resultado 1 si y sólo si ambos operandos son 1. Cuando utilizamos este operador, lo que estamos diciendo es que el resultado será verdadero cuando todos los operandos sean verdaderos. Compárese con el producto aritmético para ver por qué este también se denomina *producto*.

Aún nos quedan dos operaciones lógicas bastante utilizadas. La operación lógica NOT se aplica sobre un único operando y tiene el mismo efecto que el complemento a 1, es decir, cuando el operando es 0, la operación devuelve 1, y cuando el operando es 1, devuelve 0.

La última operación lógica que vemos aquí es la suma lógica exclusiva o XOR cuyo funcionamiento es análogo al operador OR excepto en que cuando los dos operadores son 1, el resultado es cero; de ahí el término de *exclusiva*.

Suma lógica exclusiva: XOR

a	b	a XOR b
0	0	0
0	1	1
1	0	1
1	1	0

Negación lógica: NOT

a	NOT a
0	1
1	0

Estas operaciones lógicas nos serán útiles cuando tengamos que programar diversos problemas en los que es preciso comprobar si determinadas condiciones son verdaderas o falsas a partir de otras proposiciones lógicas. Por

ejemplo, para saber si un año es bisiesto, es preciso comprobar si es divisible por 4 pero no por 100, o si es divisible por 400. Podríamos utilizar tres variables lógicas, a partir de las cuales se calcula la condición Z que valdrá 1 si el año es bisiesto o 0 en caso contrario:

a = el año es divisible por 4

b = el año es divisible por 100

c = el año es divisible por 400

Z = *a* AND NOT *b* OR *c*

La forma de plantearlo ha sido "Z es verdadero si y sólo si *a* es verdadero Y *b* NO es verdadero O *c* es verdadero". En el capítulo de algorítmica veremos la precedencia de estos operadores; en este punto simplemente digamos que el orden de precedencia de los operadores utilizados es NOT – AND – OR.

Veamos finalmente algunos ejemplos en los que se aplican estas operaciones lógicas a números binarios de 7 bits.

Ejemplo

Realizar la operación 1001101 OR 1100101.

Esta operación es realmente sencilla, simplemente pondremos como resultado siempre 1 excepto cuando los dos bits correspondientes sean cero, en cuyo caso el resultado es 0.

```
      1 0 0 1 1 0 1
OR    1 1 0 0 1 0 1
    ---------------
      1 1 0 1 1 0 1
```

o

Ejemplo

Realizar la operación 1001101 AND 1100101.

También es muy sencilla ya que el resultado siempre será 0 excepto cuando los dos bits correspondientes sean 1 en cuyo caso el resultado será 1.

```
      1 0 0 1 1 0 1
AND   1 1 0 0 1 0 1
    ---------------
      1 0 0 0 1 0 1
```

o

Ejemplo

Realizar la operación 1001101 XOR 1100101.

Es como la que realizamos con el OR excepto que cuando ambos bits sean 1, el resultado es 0.

```
        1 0 0 1 1 0 1
XOR     1 1 0 0 1 0 1
      ---------------
        0 1 0 1 0 0 0
```

o

Base octal

Este sistema de numeración es uno de los dos denominados **sistemas intermedios**. Los sistemas de numeración intermedios (octal y hexadecimal) se utilizan a menudo en Informática por su capacidad de condensación de números binarios que de otra manera serían difíciles de manejar debido a su longitud. La característica fundamental de estos sistemas de numeración es que la base *b* es una potencia de dos.

En la base octal, $b=8=2^3$ con lo que el alfabeto en esta base es el formado por el conjunto {0, 1, 2, 3, 4, 5, 6, 7}. Dado que con estos ocho caracteres se pueden representar todas las combinaciones binarias de 3 bits, desde el 000 al 111, la conversión de binario a octal o de octal a binario es inmediata. Simplemente, para pasar de **binario a octal**, se cogen grupos de tres bits partiendo de la derecha para la parte entera y desde la izquierda para la parte fraccionaria estableciendo el equivalente de cada grupo en octal. Estas equivalencias son como las mostradas en la tabla de la página 28.

Ejemplo

Convertir el número binario 10001101100,110100 a octal.

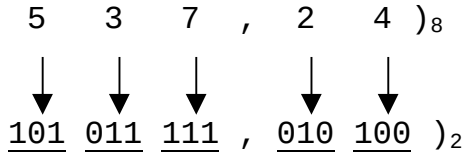
```
10 001 101 100 , 110 100 )2
 ↓   ↓   ↓   ↓     ↓   ↓
 2   1   5   4  ,  6   4 )8
```

o

Para pasar de **octal a binario**, se vuelve a usar la misma tabla y cada cifra en octal se convierte directamente en 3 bits.

Ejemplo

Convertir el número octal 537,24 a binario natural.



o

Base hexadecimal

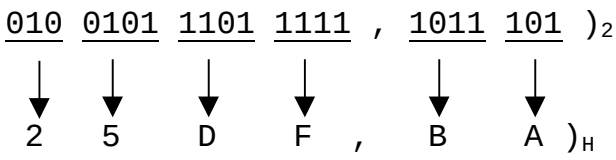
El otro sistema de numeración intermedio es el sistema hexadecimal. Este sistema de numeración es muy común en Informática; incluso más que el octal. Este sistema permite representar la información de forma más compacta que el octal. En este sistema de numeración, como su propio nombre indica, la base es $b=16=2^4$, es decir, cada 4 bits son representables por una sola cifra en este sistema de numeración. El alfabeto utilizado está formado por los 16 caracteres siguientes: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}.

A la hora de manejar este sistema de numeración y poder realizar conversiones a / de decimal o binario natural, conviene recordar la siguiente tabla:

Hex.	Dec.	Bin.	Hex.	Dec.	Bin.
0	0	0000	8	8	1000
1	1	0001	9	9	1001
2	2	0010	A	10	1010
3	3	0011	B	11	1011
4	4	0100	C	12	1100
5	5	0101	D	13	1101
6	6	0110	E	14	1110
7	7	0111	F	15	1111

Ejemplo

Convertir a hexadecimal el número binario 0100101110111111,1011101.



De forma análoga al caso octal, cogemos, en este caso, grupos de 4 bits y los sustituimos por su equivalente en hexadecimal. o

Fíjese que tanto en octal como en hexadecimal, en caso en que el primer grupo de la parte entera no conste de 3 o 4 bits, respectivamente, se puede rellenar con ceros por la izquierda, con lo que el número no varía. De forma análoga, el último grupo de la parte fraccionaria debe completarse con ceros por la derecha para disponer de un grupo de 3 bits (octal) o 4 bits (hexadecimal).

Por otra parte, dijimos anteriormente que para hacer referencia al sistema de numeración en el que se daba un número se añadía a su derecha el símbolo de cerrar paréntesis y la base como subíndice, de manera que pudiéramos distinguir, por ejemplo, el $111)_2$ del $111)_{10}$. Pues bien, en el sistema de numeración octal es habitual emplear tanto el 8 como la letra 'O'. De la misma manera, en el sistema hexadecimal se utiliza tanto el 16 como la letra 'H', como se ha hecho en el ejemplo anterior.

2.2.2 Códigos de Entrada / Salida

Comenzábamos esta parte de representación de la información realizando unas consideraciones sobre cómo iba a representarse esta información en el ordenador. Decíamos que esta información siempre se almacena en forma de ceros y unos, pero que puede estar almacenado como un código binario (representación interna numérica) o como un código de entrada / salida. En este último caso, cada secuencia de ceros y unos de una determinada longitud representará un carácter o símbolo. Por lo tanto, podemos asumir que los caracteres se almacenan como una combinación de bits (por ejemplo, este texto se escribió en un ordenador y se almacenaba de esa forma).

Dado un conjunto de caracteres finito que nosotros queramos que se corresponda con un código de entrada / salida en el ordenador, podemos preguntarnos cuantos bits necesitamos. O podemos empezar planteando la pregunta a la inversa: dados un número determinado de bits, cuántos caracteres podemos representar, o lo que es lo mismo, cuántas combinaciones podemos formar. Esto ya lo sabemos hacer:

2 bits $\rightarrow$ 4 combinaciones

3 bits $\rightarrow$ 8 combinaciones

n bits $\rightarrow 2^n$ combinaciones

Por lo tanto, con *n* bits podemos representar *m* símbolos, con $m = 2^n$. Si queremos representar *m* símbolos y queremos saber el número de bits necesario, solamente tenemos que despejar n de la ecuación anterior: $n = log_2 m$. Pero, como *n* tiene que ser entero, la forma de obtener el número de bits necesario para representar *m* símbolos es:

$$n \geq \log_2 m$$
$$n \in \aleph$$

Existen varios códigos de entrada / salida estándar. Es decir, no nos tenemos que "inventar" nosotros el código de entrada / salida, lo cual sería mala idea ya que luego tendríamos problemas a la hora de intercambiar información con otros ordenadores. El código de entrada / salida más ampliamente utilizado y el más conocido es el código ASCII (Apéndice A), cuyas iniciales provienen del inglés (American Standard Code for Information Interchange). Este código está formado por 7 bits, por lo que sólo se pueden representar 128 símbolos distintos. Fíjese también que, en dicha tabla, los primeros símbolos son los denominados caracteres de control, que son caracteres no representables pero que tienen significado para el ordenador a la hora de representar un texto, como pueden ser los caracteres de tabulación, nueva línea, retorno de carro, fin de fichero, etc. Por otra parte, si prescindimos de estos caracteres no representables, con 7 bits no nos quedan muchos más símbolos que representar, en particular, no aparecen ni las vocales acentuadas, ni la 'eñe', para el caso castellano, así como otros caracteres de otros lenguajes como el alemán, francés, finlandés, etc. Es por ello, que a menudo se utiliza el denominado ASCII extendido que consta de 8 bits con lo que se pueden representar 128 caracteres más, en particular todos los símbolos internaciones adicionales que comentábamos hace un momento.

Finalmente, respecto a otros códigos de entrada / salida distintos del ASCII, en su día tuvo su importancia el código BCD de intercambio normalizado (6 bits, Standard Binary Coded Decimal Interchange Code) y el EBCDIC (8 bits, Extended Binary Coded Decimal Interchange Code).

2.2.3 Detección de errores en la información codificada

Hoy día, es habitual que la información no se quede "aislada" dentro de un ordenador, sino que la información se suele transmitir de unos ordenadores a otros. El medio de transmisión puede ser múltiple (fibra óptica, pares de cobre, coaxial, radiofrecuencia, etc.). No vamos a entrar en los detalles; lo que nos interesa de este hecho es que cuando se transmite la información por alguno de estos medios, existe una probabilidad de que lo que se transmite no sea exactamente igual a lo que se recibe. La transmisión de la información puede ir modulada de varias formas. Lo habitual es utilizar alguna técnica de modulación. Esta técnica no es objeto del presente texto, pero para hacernos una idea, pensemos en un aparato de radio: podemos escuchar emisoras en FM

(se recibe la señal modulada en frecuencia) o en AM (modulación en amplitud). Nuestro objetivo ahora es un poco más sencillo, vamos a suponer que representamos los ceros como niveles de voltaje bajos y los unos como niveles de voltaje altos. Es perfectamente comprensible que una interferencia que afecte al medio de transmisión puede provocar que un cero se convierta en un uno o que un uno se convierta en un cero. Es importante darse cuenta que estos efectos, aunque ocurran, suelen verse minimizados por mecanismos de corrección de errores o al utilizar medios más fiables, es decir, son poco probables en la práctica, excepto en ambientes con muchas interferencias como puede ser un entorno industrial. Por lo tanto, aunque sea poco probable, necesitamos algún método sencillo que nos permita detectar cuándo la información ha sufrido modificación en su tránsito del emisor al receptor. Sin embargo, antes de tratar este tema, debemos comentar algunas definiciones como eficiencia de un código de entrada / salida y redundancia.

En el apartado anterior comentábamos que si queríamos representar *m* símbolos, necesitábamos $n \geq log_2\ m$ bits, con *n* el menor entero mayor que dicho logaritmo. Es decir, que si necesitamos representar, por ejemplo, 50 símbolos, es fácil deducir que el número de bits necesarios es al menos 6. Sin embargo, con 6 bits, podríamos representar hasta 64 símbolos, por lo que es obvio que existen combinaciones de ceros y unos que no se corresponden con ningún carácter. A partir de este hecho, podemos definir la **eficiencia** de un código, τ, como:

$$\tau = \frac{m}{2^n} \qquad 0 \leq \tau \leq 1$$

donde *m* es el número de símbolos que queremos representar y *n* el número de bits que utilizamos.

Ejemplo

Calcular la eficiencia de utilizar el código ASCII para representar 95 símbolos.

Como el código ASCII consta de 7 bits, la eficiencia la calculamos aplicando la fórmula vista anteriormente:

$$\tau = \frac{95}{2^7} = 0{,}742$$

o

Hablar de que un código es poco eficiente (en el sentido de que la eficiencia tiene un valor bajo), es lo mismo que decir que es un código redundante (valor alto de la redundancia según se define a continuación). La **redundancia** es otro índice función de la eficiencia y que se define como:

$$R = (1 - \tau) \times 100$$

Para el ejemplo anterior, la redundancia se calcularía como R = (1-0.742)·100 = 25,8%.

¿Y para qué sirve un código redundante? Pues sirve para detectar posibles errores en el código que nos llega. En este punto, para que se entienda de forma fácil, supóngase el caso del ejemplo donde sólo representamos 95 símbolos. En ese caso, existen 128 – 95 = 33 combinaciones de ceros y unos que no simbolizan nada. Por lo tanto, si en una transmisión de información utilizando dicho código, nos llega alguna de esas 33 combinaciones, podemos concluir que ha habido un fallo en la transmisión (asumiendo que en el otro extremo utilizan el mismo código y que las combinaciones "prohibidas" no se generan).

Existen diversas pautas para hacer un código redundante y poder detectar errores. El más conocido es el denominado **bit de paridad**. Este método consta a su vez de dos criterios (par e impar).

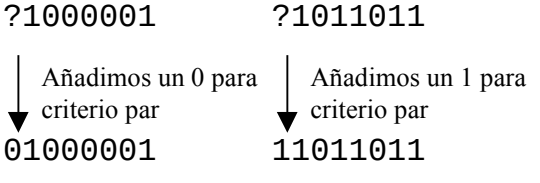
> Bit de paridad, criterio par: "se añade un bit tal que el número total de unos sea par"

El bit de paridad se añade a cada carácter antes de la transmisión, añadiendo un 1 o un 0 en la posición más significativa (a la izquierda) del código a transmitir. Por ejemplo, al código ASCII de 7 bits se le puede añadir el bit de paridad para transmitir cierta información, con lo que por cada símbolo se envían 8 bits en lugar de 7. Cuando esta información se recibe en el otro extremo de la comunicación, se analiza lo recibido para comprobar que en cada símbolo el número total de unos es par. Si en algún símbolo, el número total de unos es impar, significa que se ha cambiado un cero por un uno o viceversa.

Ejemplo

Generar el código con bit de paridad criterio par para 1000001 y 1011011.

Estamos trabajando con un código de 7 bits, por lo que al añadir el bit de paridad en la posición 7, se convertirá en un código de 8 bits. El código 1000001 tiene 2 unos, es decir, ya existe un número par de unos, por lo que para que se mantenga la paridad debemos añadir un 0. Para el otro código, 1011011, tiene 5 unos, es decir, para que haya un número par de unos debemos añadir un 1, con lo que ya habría 6 unos.



o

Este método de detectar errores nos permite darnos cuenta de un error en la transmisión cuando cambia un único bit en alguno de los caracteres transmitidos. En la mayoría de los casos esto es suficiente, ya que si por ejemplo, la probabilidad de cambio en un bit es de 10^{-6}, la probabilidad de que cambien dos bits simultáneamente sería de 10^{-12}, lo cual podemos considerar altamente improbable. Además, en general, se combina el uso de la paridad con otras técnicas de control de errores.

Existen también métodos que permiten, no solamente detectar cuándo ha sucedido un error, sino en determinados casos, corregirlo. En el esquema mostrado, la única alternativa cuando se detecta un error es pedir al transmisor que vuelva a reenviar la información ya que no sabemos cuál de los bits ha cambiado para reconstruir el código original.

2.2.4 Representación interna de la información

Los códigos de entrada / salida son una de las formas de almacenar la información en el ordenador. Para almacenar texto suele ser bastante eficiente, al menos si la redundancia del código es pequeña. Sin embargo, cuando se trata de almacenar valores numéricos, esta forma de codificación es muy poco eficiente.

Imaginemos que deseamos almacenar el número $255)_{10}$. Si utilizáramos un código ASCII extendido de 8 bits, necesitaríamos utilizar un espacio de almacenamiento de 3 caracteres * 8 bits / carácter = 24 bits = 3 bytes. Sin embargo, ahora sabemos que el 255 en decimal puede expresarse como 11111111 en binario, es decir, que si pudiéramos almacenar esta información como número en binario, solamente ocuparíamos 8 bits = 1 byte, es decir, 3 veces menos.

A la hora de representar información numérica internamente, es necesario distinguir entre los distintos tipos de datos numéricos con el objetivo de optimizar más aun el espacio de almacenamiento. Aunque hoy en día, esta necesidad de "ahorrar" memoria puede parecer innecesaria, hay que tener en cuenta que las bases de la Informática se establecieron cuando los recursos en cuanto a memoria eran escasos. Además, el hecho de almacenar la información de manera mucho más eficiente significa programas que se ejecutan más rápidamente. Vamos a distinguir, por lo tanto, algunos tipos básicos de datos cuyo almacenamiento interno se realiza de forma distinta: **enteros sin signo**, **enteros con signo** y **números en punto flotante**.

Las bases sobre las que vamos a trabajar son las siguientes: se considera un dato numérico N el cual debemos almacenar en *n* posiciones (o bits). Además usaremos la siguiente nomenclatura:

> Bit más significativo o MSB[4]: "se refiere a la posición a_{n-1} de un número binario de *n* bits, es decir, la posición más a la izquierda del número"

> Bit menos significativo o LSB[5]: "se refiere a la posición a_0 de un número binario, es decir, la posición más a la derecha del número"

Enteros sin signo

Los números enteros sin signo se almacenan en forma de binario natural. Normalmente se nos indicará el espacio de almacenamiento que tenemos disponible en cuanto a número de bits. Por ejemplo, si nos dicen que el espacio de almacenamiento es de 8 bits, significa que podemos almacenar desde el $0)_{10}=00000000)_2$ hasta el $255)_{10}=11111111)_2$.

Enteros con signo

Cuando nos planteamos el almacenamiento de números enteros con signo, es decir, que pueden ser tanto positivos como negativos, caben varias posibilidades. Podríamos reservar el bit más significativo para el signo, es decir, que cuando el número sea positivo, ponemos en esa posición un 0, y cuando sea negativo, ponemos un 1. También podríamos hacer uso de lo que hemos aprendido de los complementos (página 31) y utilizarlos cuando el número sea negativo. De esta forma, en la representación de enteros con signo disponemos de varias posibilidades de almacenamiento:

- Signo y magnitud
- Complemento a 1
- Complemento a 2
- Entero sesgado

En la representación en **signo y magnitud**, como dijimos antes, el bit más significativo se reserva para el signo y el resto de los bits para la magnitud del número en binario natural; de esta manera, se obtendría la representación que se muestra en la Figura 2-3.

---

[4] Most Significant Bit

[5] Less Significant Bit

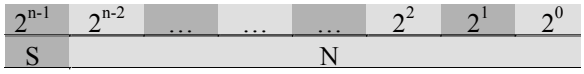


Figura 2-3. Representación en signo y magnitud

En cuanto a las representaciones en **complemento a 1** y **complemento a 2**, en caso de que el número a representar sea **positivo**, se utiliza el mismo criterio que en el caso anterior, es decir, el MSB para el signo y el resto de las posiciones para la magnitud del número. En caso de que el número sea **negativo**, la representación interna pasa a ser el complemento a 1 ó a 2 (según se trate) del valor absoluto del número.

En estas representaciones, el mayor número positivo representable es $a = 2^{n-1} - 1$ como es fácilmente demostrable.

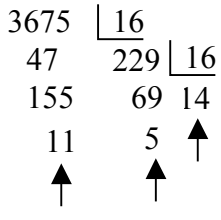
En la representación en **entero sesgado**, se define el denominado **sesgo** $S = 2^{n-1}$ que se suma al valor del número a representar, siendo el resultado, en binario natural, la representación interna a almacenar.

Con el objetivo de clarificar aún más estas representaciones, vamos a mostrar un ejemplo.

Ejemplo

Obtener la representación interna del número entero –3675 considerando que el espacio de almacenamiento es de 2 bytes. Calcular dicha representación en signo y magnitud, complemento a 1, complemento a 2 y entero sesgado.

En primer lugar, obtenemos la representación en binario de la magnitud del número: 3675, para lo cual habría que ir dividiendo por 2 sucesivamente. Evidentemente, ante la magnitud de dicho número, esta conversión puede hacerse muy tediosa; por ello, vamos a hacer uso de uno de los sistemas de numeración intermedios, el hexadecimal, de manera que pasaremos el número a hexadecimal y de ahí, el pasarlo a binario es inmediato.



$14)_{10} = E)_H$, $5)_{10} = 5)_H$, $11)_{10} = B)_H$, por lo que $3675)_{10} = 0E5B)_H = 0000\ 1110\ 0101\ 1011)_2$ donde hemos rellenado con ceros por la izquierda para "ocupar" los dos bytes que tenemos reservados.

a) *Signo y magnitud: el número es negativo por lo que el MSB $a_{n-1} = 1$ y la representación interna es 1000 1110 0101 1011, que en hexadecimal es 8E5B.*

b) *Complemento a 1: el complemento a 1 de 0E5B lo podemos hacer directamente en hexadecimal sin más que realizar la operación FFFF – 0E5B = F1A4, por lo que la representación interna es 1111 0001 1010 0100.*

c) *Complemento a 2: simplemente tenemos que sumar 1 al complemento a 1, F1A4 + 0001 = F1A5, por lo que la representación interna es 1111 0001 1010 0101.*

d) *Entero sesgado: el sesgo es* $S = 2^{n-1} = 2^{16-1} = 32768$*, así que para calcular la representación interna debemos sumar el sesgo y el número* $32768 + (-3675) = 29093)_{10} = 71A5)_H$ *por lo que la representación interna es 0111 0001 1010 0101.*

Tipo real

A la hora de representar un número de tipo real, es conveniente utilizar la denominada notación exponencial o notación científica o notación en punto flotante o notación en coma flotante, por todos estos nombres se es conocida. La ventaja de esta notación es que podemos separar lo que es el orden de magnitud del número y el valor del número con gran precisión, todo ello de forma bastante compacta. En la notación exponencial, muy utilizada en ámbitos científicos, se separa el número en mantisa y exponente. Lo que se hace es representar el número de la forma $M \cdot B^E$ donde M es la mantisa, B la base del sistema de numeración utilizado y E el exponente. De esta manera se pueden representar de forma adecuada tanto números muy grandes como muy pequeños sin más que considerarlo en el exponente. Así, se tiene que el número de Avogadro es $6{,}022 \times 10^{23}$ partículas / mol en notación exponencial; si lo representáramos con todas las cifras (notación en punto fijo) deberíamos escribir 602200000000000000000000,0 partículas / mol, asumiendo la precisión dada originalmente. De la misma manera, para números muy pequeños como la carga fundamental $1{,}602 \times 10^{-19}$ C, la notación en punto fijo sería 0,0000000000000000001602 C.

La notación en punto flotante es más adecuada y compacta para representar este tipo de valores de amplio uso en las diversas ramas científicas. Se denomina de punto flotante porque la coma decimal, o punto para los anglosajones, es flotante, es decir, se puede desplazar hacia la izquierda o hacia la derecha, y el número de posiciones que la desplacemos se ve reflejado en el exponente; esto es así porque desplazar la coma decimal una posición a la derecha es lo mismo que multiplicar por 10, por lo que el exponente se vería decrementado en uno para que el número no varíe: $0{,}15 = 1{,}5 \times 10^{-1}$. De la misma manera, desplazar la coma decimal una posición a la izquierda es lo mismo que dividir por 10, por lo que el exponente se vería incrementado en uno: $1{,}5 = 0{,}15 \times 10^{1}$. Esto que hemos indicado es cierto en el sistema de numeración decimal (base $b = 10$), de ahí que el efecto de desplazar la coma

decimal se vea reflejado en el exponente de 10. En otros sistemas de numeración, el desplazar la coma a izquierda o derecha significa dividir o multiplicar por la base.

Sin embargo, como se puede uno imaginar, nosotros estamos interesados en la representación en base 2, lo que da lugar a la denominada **Normalización IEEE 754**, un estándar de representación de números en punto flotante en el ordenador. En este estándar, se supone que la base predeterminada en la representación es B = 2, lo cual supondremos de aquí en adelante. En la representación $N = M \cdot B^E$ la mantisa contiene el signo del número y el signo del exponente simplemente indica las posiciones a izquierda o derecha que se debe desplazar la coma decimal. En la representación IEEE 754, el signo del número se considera como un elemento aparte, por lo que se almacenan 3 campos: campo del signo *s*, campo del exponente *e* y campo de la mantisa *m* (Figura 2-4).

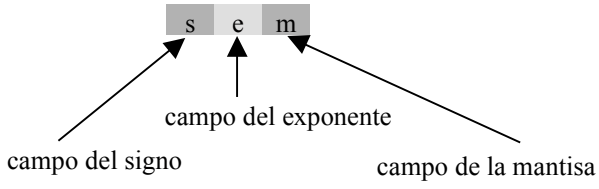


Figura 2-4. Campos en el estándar IEEE 754

Destaquemos que hablamos de campos, no de que almacenemos de esa forma directamente la mantisa y el exponente; ahora veremos cómo se calculan estos campos. En primer lugar, al ser una representación en el ordenador, todos esos campos constarán de ceros y unos. ¿Cuánto espacio se necesita para cada campo? Para el campo del signo *s*, con un bit es suficiente, que valdrá cero si el número es positivo o uno si es negativo. Para el campo del exponente (incluye el signo de este) supondremos que tenemos reservados *ne* bits y para el campo de la mantisa *nm* bits. Por lo tanto, el número total de bits almacenados será $n = 1 + ne + nm$. Pasemos a continuación a ver cómo se calculan cada uno de los campos (el del signo ya lo hemos visto).

El **campo del exponente** se almacena en forma de entero sesgado, que ya vimos cuando representábamos números enteros. Recordemos que se suma, en este caso, el exponente E más el sesgo S, calculado como $S = 2^{ne-1} - 1$. Es decir, el campo del exponente se calcula como

$$e = S + E = 2^{ne-1} + E - 1$$

Si, por ejemplo, el número de bits del campo del exponente es $ne = 8$ bits, el sesgo es $S = 2^{8-1} - 1 = 127 = 01111111)_2$. De esta manera, si el exponente que queremos almacenar es 0, el exponente sesgado sería 127 + 0 = 127 y el campo del exponente 01111111; si el exponente es +2, el exponente sesgado es 127 + 2 = 129 y el campo del exponente 10000001; si el exponente es –126, el exponente sesgado es 127 + (-126) = 1 y el campo del exponente 00000001.

Nótese que el número de bits del campo del exponente determina el que podamos almacenar números de magnitud muy grande o muy pequeña; en el ejemplo del párrafo anterior, la magnitud más pequeña que podíamos almacenar era 2^{-127} y la más grande 2^{128} (realmente es 2^{-126} y 2^{127}, ya que los casos extremos son casos especiales como veremos más adelante).

Para almacenar el **campo de la mantisa**, es preciso previamente normalizar el número. Esta normalización se debe hacer también antes de almacenar el campo del exponente y consiste en que la mantisa debe aparecer con un 1 más significativo en la posición 0 teniendo en cuenta que en el campo de la mantisa no se almacena dicho 1, solamente la parte fraccionaria. Es decir, debemos conseguir que la mantisa esté limitada $1 \leq M < 2$, o lo que es lo mismo, la relación entre mantisa M y campo de la mantisa m es M = [1,m].

Para clarificar de manera suficiente este tipo de representación pondremos un par de ejemplos.

Ejemplo

Obtener la representación interna en formato IEEE 754 en simple precisión (número de bits: 32, número de bits del exponente: 8) del número -37,25.

Como el número total de bits son 32 y tenemos 8 para el exponente y 1 para el signo, nos quedan 32 – 8 – 1 = 23 bits para la mantisa. Los pasos que debemos dar son los siguientes:

- *Pasar el número de decimal a binario: $-37{,}25)_{10} = -100101{,}01)_2$*
- *Normalizarlo, es decir, conseguir que la mantisa quede como 1,… para lo que añadiremos un exponente de 2: $-100101{,}01 = -1{,}0010101 \times 2^5$. Identificamos **m = 0010101**… y E = 5*
- *Calcular el sesgo (número de bits para el exponente ne = 8) $S = 2^{8-1} – 1 = 2^7 – 1 = 127$*
- *Calcular el campo del exponente **e** = E + S = 5 + 127 = $132)_{10} =$ **10000100**$)_2$*

Ahora debemos completar la representación con los tres campos: s = 1, m = 0010101… y e = 10000100 teniendo en cuenta que para la mantisa debemos rellenar con ceros por la derecha hasta completar los 23 bits. Para el exponente, ya tenemos 8 bits, pero si tuviéramos menos completaríamos con ceros por la izquierda. ¡Razonar por qué!. Finalmente, la representación interna queda:

1 10000100 00101010000000000000000 o

Planteemos ahora el caso inverso, es decir, dada la representación interna de un número real en el ordenador, obtener a qué número corresponde.

Ejemplo

Obtener el número real a que corresponde la siguiente representación interna: 1 0011 1110 0011 110 sabiendo que el número de bits total *n* es 16 y el número de bits para el exponente es *ne = 8*.

El bit más significativo es 1 que corresponde al signo, luego ya sabemos que el número es negativo.

A continuación viene el campo del exponente que serán los 8 bits siguientes: 0011 1110 = 62)$_{10}$. El sesgo es, al igual que en el ejemplo anterior, S = 127. Como e = E + S, despejamos E = e – S = 62 – 127 = - 65.

Por último, el campo de la mantisa son los bits restantes m = 0011 110. La mantisa se forma como M = 1,0011110)$_2$ = 1 + 2^{-3} + 2^{-4} + 2^{-5} + 2^{-6} = 1,234375.

Finalmente, conociendo el signo, la mantisa M y el exponente E, formamos el número N = -M·2^E = -1,234375·2^{-65} = -3,345780142·10^{-20}. o

En el primer ejemplo, hablábamos de formato IEEE 754 en simple precisión y que hacía referencia al número de bits en mantisa y exponente que se usaban. Dependiendo de la aplicación, si trabajamos con números muy grandes, es posible que con 8 bits en el exponente no sea suficiente. Si necesitamos mucha precisión en cuanto a número de decimales significativos, es posible que con 23 bits para la mantisa tampoco sea suficiente. Para normalizar las distintas variantes de número de bits en mantisa y exponente, el estándar IEEE 754 propone varios tipos de números reales: simple precisión, doble precisión, etc. según la tabla siguiente:

	Precisión			
	Simple	Simple ampliada	Doble	Doble ampliada
nm	24	≥ 32	53	≥ 64
$E_{máx}$	127	≥ 1023	1023	≥ 16383
$E_{mín}$	-126	≤ -1022	-1022	≤ -16382
S	127	n.e.	1023	n.e.

nm (bits de precisión), $E_{máx}$ (exponente máximo), $E_{mín}$ (exponente mínimo), S (sesgo del exponente), n.e. (no especificado)

Existen ciertos casos especiales en la representación IEEE 754 que hay que tener en cuenta:

1. Si el campo del exponente vale cero (*e = 0*), la mantisa se almacena sin normalizar y el sesgo vale $S = 2^{ne-1}-2$ por lo que el exponente del número real valdría $E = e - S = -2^{ne-1}+2$.
2. Para almacenar el número real *N=0*, se representa con todos los bits igual a cero (*e = 0, m = 0*).
3. En el caso de que todos los bits del campo del exponente valgan 1, el dato está representando $\pm\infty$ en caso de que la mantisa valga cero o NaN[6] en caso de que la mantisa sea distinta de cero.

Precisión y redondeo

Las operaciones matemáticas que realiza el ordenador carecen en algunas ocasiones de algunas propiedades deseables como la asociativa y distributiva. Ello es debido a que en un intervalo dado de la recta real existen infinitos valores reales y la representación computacional tiene una precisión finita por lo que en muchos cálculos es posible que se pierdan bits que no "caben" en la longitud asignada por la representación. Esto puede sorprender en un principio a una persona de formación científica, pero es de vital importancia, especialmente, en la programación científica, ya que si no lo tenemos en cuenta podemos llegar a unos resultados erróneos. En particular, hay que prestar especial atención a tres operaciones que pueden causar que el programa no funcione de forma lógica:

1. Al restar dos números muy parecidos o al dividir un número por otro mucho mayor, obtendremos números *excesivamente pequeños*, por lo que podemos perder precisión o incluso provocar un desbordamiento a cero (*underflow*).
2. Al dividir un número por otro mucho menor o al realizar sumas o productos sucesivos con valores grandes, obtendremos números *excesivamente grandes*, por lo que podemos provocar un desbordamiento (*overflow*).
3. Al comparar dos números reales para ver si son iguales, hay que tener en cuenta que el ordenador detectará la igualdad si y sólo si tienen iguales *todos* sus bits, por lo que en estos casos es mejor realizar la comparación de igualdad con números enteros o comprobando si la diferencia entre los dos es menor que un valor suficientemente pequeño.

[6] NaN viene del inglés "Not A Number" y representa valores no válidos como puede ser $0\cdot\infty$, la raíz cuadrado de un número negativo, etc.

2.3 Lógica binaria

En lo que llevamos estudiado del ordenador hemos visto que éste trabaja con dos valores discretos, que son el cero y el uno. Decimos que tal sistema es un **sistema binario**. En la práctica, dentro de la circuitería del ordenador, dichos valores en realidad son dos niveles de potencial. Estos niveles de potencial pueden fluctuar y se considera que un potencial entre 2,4 y 5 V equivale a un "1" lógico y que un potencial entre 0 y 0,8 V equivale a un "0" lógico, según la Figura 2-5.

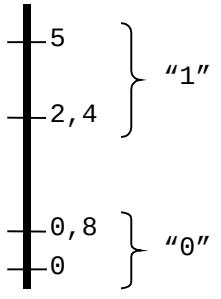


Figura 2-5. Equivalencia entre nivel de potencial y valor lógico

La razón de que los valores lógicos estén definidos para un intervalo de voltajes se debe a razones de seguridad para que aunque haya pérdidas de potencial en los diversos circuitos por los que transitan las señales, si éstas son pequeñas, se siga manteniendo el valor lógico.

Los **sistemas o dispositivos digitales** se caracterizan por responder con salidas binarias ante un determinado número de entradas binarias (Figura 2-6).

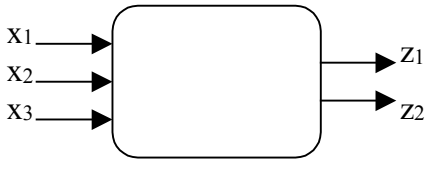


Figura 2-6. Representación de un sistema digital

Esta es una representación abstracta y no dice mucho de las operaciones que realiza dicho sistema. Una forma de caracterizar a un sistema digital como el de la figura es utilizando la denominada **tabla de verdad**, que consiste en obtener todas las posibles salidas en respuesta a toda combinación de entradas; algo semejante a lo que hacíamos cuando veíamos en la sección 2.2.1 las operaciones lógicas AND, OR, … sólo que en este caso tenemos 3 entradas y 2 salidas, con lo que la tabla de verdad podría ser la siguiente:

	x_1	x_2	x_3	z_1	z_2

0	0	0	0	0	0
1	0	0	1	0	1
2	0	1	0	1	1
3	0	1	1	0	0
4	1	0	0	1	1
5	1	0	1	0	1
6	1	1	0	0	0
7	1	1	1	0	0

Como se ve en la tabla, todas las posibles combinaciones de las entradas corresponden a las combinaciones binarias desde el 000 hasta el 111; para facilitar su comprensión se adjunta a su izquierda el valor decimal. Para cada una de estas combinaciones se añade los valores de z_1 y z_2.

Es decir, que a diferencia de los sistemas analógicos en los que una función habría que definirla para infinitos valores (si la quisiéramos dar en forma de tabla), en los sistemas digitales, al trabajar con valores discretos (en particular binarios), se puede definir una función dando todas las posibles combinaciones.

Un sistema **digital** describe cualquier sistema basado en datos discontinuos o eventos. Los ordenadores son sistemas digitales ya que, en el nivel más básico sólo distingue dos valores, 0 y 1, o apagado y encendido. Todo dato que el ordenador procesa debe ser codificado digitalmente, como una serie de ceros y unos. Lo contrario de digital es **analógico**. Un dispositivo analógico típico es un reloj en el que las agujas se mueven continuamente de forma circular. Un reloj como el descrito es capaz de indicar cualquier momento del día. En contraposición, un reloj digital sólo es capaz de representar un número finito de instantes (décimas de segundo, por ejemplo). Véase Figura 2-1. Internamente, los ordenadores son digitales porque trabajan con unidades elementales llamadas bits, como ya sabemos, mas combinando muchos bits, son capaces de simular eventos analógicos.

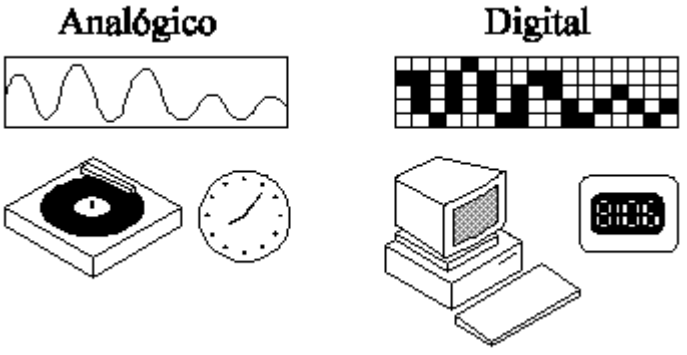


Figura 2-7. Analógico frente a Digital

En este punto, nos podemos preguntar qué nos va a aportar el estudio de la lógica binaria a la programación científica. A la hora de realizar un programa, no solamente vamos a instruir al ordenador para realizar operaciones sino que según sea el resultado de ciertas operaciones o en función de determinados datos, es posible que el programa deba comportarse de manera distinta. Es en este punto donde entra la lógica y donde será necesario, en muchos casos, aplicar los conocimientos cuyas bases vamos a dar a continuación.

2.3.1 Álgebra de Boole

El álgebra de Boole la podemos considerar como una herramienta matemática idónea para análisis y síntesis de circuitos digitales. Nuestro interés se centrará en aquellas propiedades que nos permitan resolver problemas computacionalmente.

Para definir un **álgebra**, es necesario disponer de tres elementos:

- Un conjunto de elementos sobre los que se aplica el álgebra
- Operaciones binarias[7] o leyes de composición interna
- Principios básicos o axiomas

En particular, el álgebra de Boole va a estar definido como una estructura <B, +, ·, ¯>, que significa que es una estructura que se aplica a un conjunto de elementos B, y que las operaciones definidas son la suma +, la multiplicación ·, y el opuesto ¯. Además debemos definir los **postulados** del álgebra de Boole que son:

I. Conjunto de elementos: $\exists B / x, y \in B, x \neq y$

II. Leyes de composición interna: $\forall x, y \in B, x \cdot y \in B, x + y \in B$

III. Elementos neutros únicos:

[7] Aquí operaciones binarias se refiere a operaciones que se efectúan sobre dos elementos

a) $\exists 0! \in B / \forall x \in B, x + 0 = 0 + x = x$

b) $\exists 1! \in B / \forall x \in B, x \cdot 1 = 1 \cdot x = x$

c) $0 \neq 1$

IV. Conmutatividad: $\forall x, y \in B, x + y = y + x, x \cdot y = y \cdot x$

V. Distributividad: $\begin{cases} \forall x, y, z \in B, x + (y \cdot z) = (x + y) \cdot (x + z) \\ x \cdot (y + z) = (x \cdot y) + (x \cdot z) \end{cases}$

VI. Asociatividad: $\begin{cases} \forall x, y, z \in B, x + (y + z) = (x + y) + z = x + y + z \\ x \cdot (y \cdot z) = (x \cdot y) \cdot z = x \cdot y \cdot z \end{cases}$

VII. Elemento opuesto único: $\forall x \in B, \exists \overline{x}! \in B / x + \overline{x} = 1, x \cdot \overline{x} = 0$

El postulado I nos indica que en el conjunto de elementos podemos encontrar al menos dos que son distintos. El postulado II nos indica que el resultado de aplicar las leyes de composición interna + y · sigue perteneciendo al conjunto B. El postulado III nos indica que existe un elemento neutro para la suma al que denotamos por 0 y que és único, de la misma manera existe un elemento neutro para el producto al que denotamos por 1 y que también es único, pero además estos dos elementos deben ser distintos. Los postulados IV, V y VI nos enuncian las propiedades conmutativa, distributiva y asociativa. Finalmente, el postulado VII nos dice las propiedades que tiene el elemento opuesto y que además éste es único para cada elemento del conjunto.

Nos podemos fijar en una característica interesante de estos postulados y que se denomina propiedad de **dualidad**: los enunciados vistos, en general, aparecen por parejas donde podemos transformar una parte del enunciado en la otra sin más que sustituir los ceros por los unos y viceversa, y el operador + por el · y viceversa. Por ejemplo, en el último postulado, si a la primera parte $x + \overline{x} = 1$ le cambiamos el + por el · y el 1 por el 0, se nos convierte en la segunda parte $x \cdot \overline{x} = 0$.

Dentro del álgebra de Boole se suelen considerar varios teoremas, que se demuestran a partir de los postulados vistos; se deja propuesta su demostración al lector.

Teoremas

- Ley de idempotencia: $\forall a \in B, a + a = a, a \cdot a = a$
- $\forall a \in B, a + 1 = 1, a \cdot 0 = 0$
- $\overline{1} = 0, \overline{0} = 1$
- Ley de Morgan: $\forall a, b \in B, \overline{a + b} = \overline{a} \cdot \overline{b}, \overline{a \cdot b} = \overline{a} + \overline{b}$

- $\forall a,b \in B, a + a \cdot b = a, a \cdot (a + b) = a$
- $\forall a \in B, (\overline{\overline{a}}) = a$
- $\forall a,b \in B, a + \overline{a} \cdot b = a + b, a \cdot (\overline{a} + b) = a \cdot b$

Algebra de Boole B_2

En lo que hemos visto hasta ahora del álgebra de Boole, en ningún momento hemos indicado nada respecto a qué elementos en concreto forman el conjunto B. En Informática nos interesa trabajar con el álgebra de Boole B_2 donde el conjunto de elementos con los que trabaja es mínimo: B={0,1} y las operaciones suma, producto y opuesto se pueden definir a través de las siguientes tablas:

+	0	1
0	0	1
1	1	1

·	0	1
0	0	0
1	0	1

¯	
0	1
1	0

En dichas tablas tenemos todas las posibilidades dado el conjunto limitado de elementos de que disponemos. Las operaciones suma y producto son binarias, es decir, actúan sobre dos elementos; tomando uno de la primera fila y uno de la primera columna, en la intersección tenemos el resultado. La operación elemento opuesto es una operación unitaria, sólo actúa sobre un elemento, según el tercer teorema visto anteriormente.

Es de destacar que la operación suma es equivalente a la operación lógica OR que vimos en la sección 2.2.1, que la operación producto es equivalente a la operación lógica AND y que la operación elemento opuesto es equivalente a la operación lógica NOT, lo cual, por supuesto, ¡no es una casualidad!

Funciones de conmutación

Una función de conmutación de orden *n* es una función que establece una correspondencia entre el conjunto de n-uplas binarias y el conjunto {0,1}, es decir,

$$f : \{0,1\}^n \rightarrow \{0,1\}$$

Una función de este tipo se puede dar de dos formas, principalmente,

a) tabla de verdad

b) explícitamente

La forma de dar una función como tabla de verdad ya la vimos al comienzo de esta sección. Una función se expresa de forma explícita cuando se expresa de forma matemática utilizando variables para cada una de las entradas y los operadores del álgebra. Veamos a continuación cómo se puede pasar de una representación a otra.

Para ello, vamos a exponerlo con un ejemplo, suponiendo la función dada por la tabla de verdad de la página 51, en la que obtendremos la función de forma explícita que relaciona la salida z_2 con las entradas x_1, x_2 y x_3. Para facilitar dicha transformación, vamos a utilizar el concepto de **minterm**. Un minterm está formado por el producto de todas las variables de entrada, pudiendo aparecer el elemento opuesto de la variable en lugar de la variable. La propiedad interesante de un minterm es que vale cero si alguno de los factores vale cero. De esta manera, por ejemplo, $\overline{x}_1 \cdot \overline{x}_2 \cdot x_3$ vale "1" si y sólo si x_1=0, x_2=0 y x_3=1. Fíjese que esto corresponde con la segunda combinación de entradas en la tabla de verdad de la función y que cuando esta combinación de entradas sucede, la función z_2 debe valer 1. Por lo tanto, lo que debemos hacer es mirar para qué combinaciones de entradas la función debe valer 1, formar los minterm correspondientes y ponerlos como una suma, ya que lo que estamos diciendo es que la función vale 1 si y sólo si (x_1=0 y x_2=0 y x_3=1) o (x_1=0 y x_2=1 y x_3=0) o … En el razonamiento lógico aparece la conjunción 'y' que se simboliza con el producto o con el operador AND y la conjunción 'o' que se simboliza con la suma o con el operador OR. De esta manera, la función que buscamos es:

$$z_2 = \overline{x}_1 \cdot \overline{x}_2 \cdot x_3 + \overline{x}_1 \cdot x_2 \cdot \overline{x}_3 + x_1 \cdot \overline{x}_2 \cdot \overline{x}_3 + x_1 \cdot \overline{x}_2 \cdot x_3$$

Esta función es posible simplificarla haciendo uso de los postulados y teoremas del álgebra de Boole, en particular haciendo uso de la reducción de términos adyacentes ($A \cdot x + A \cdot \overline{x} \equiv A$) y de la propiedad de idempotencia ($A + A \equiv A$), o bien, de manera más automatizada haciendo uso de los denominados **mapas de Karnaugh**, que no veremos aquí aunque pueden verse en libros básicos de electrónica digital o en algunos libros de fundamentos de informática en los que se vea en profundidad el tema de la lógica binaria (ver la bibliografía que se adjunta). La expresión obtenida puede ser utilizada en cualquier lenguaje de programación sin más que sustituir el producto por la sintaxis que utilice el lenguaje para el operador AND y sustituir la suma por la sintaxis que utilice el lenguaje para el operador OR.

La transformación de una función dada en forma explícita a tabla de verdad es inmediata sin más que formar la tabla con todas las posibles combinaciones de las entradas y evaluar la función dada para cada combinación.

2.4 Para saber más

Existen varios libros que "cuentan" de forma más o menos clara los fundamentos de la informática. Destacaremos, de entre los que han pasado por nuestras manos, los siguientes que consideramos útiles:

- ✓ Introducción a la Informática. Prieto, A. Lloris y J.C. Torres. *McGraw Hill.*
- ✓ Fundamentos de Informática (Apuntes). Varios autores. *Departamento de Informática. Universidad de Jaén*.

2.5 Ejercicios propuestos

2.5.1 Sistemas de numeración

1. Convertir de decimal a binario:

a) 125.385 b) 74.423 c) 24.32 d) 127 e) 256

2. Transformar a decimal los siguientes números binarios:

a) 1101111 b) 1011.011 c) 110011.00101 d) 1001001

3. Efectuar las siguientes operaciones aritméticas binarias:

a) 10011+11011	b) 10111+10101	c) 1001+10001
d) 11100-10001	e) 11011-10111	f) 10000-01111
g) 10011-11000	h) 1001-11	i) 100010010-11001
j) 1101x101	k) 1011x111	l) 10110110x10110
m) 1101/101	n) 1011/111	o) 1000010001/1101

4. Efectuar las siguientes operaciones lógicas:

a) 10011 OR 11001110	b) 1001110 OR 1010
c) 11001 AND 11000110	d) 11110000 AND 101010110
e) 111 XOR 111	f) 10000 XOR 010110

5. Transformar a binario natural:

a) Hexadecimales	371	0.0AC54	90.99AB	43AC.F32
	A001			

b) Octales	372 10	0.0375	47.05407	87.234

6. Convertir en decimal:

a) Octales	7354 666	0.07612	4327.065	7777.77
b) Hexadecimales	1A90	0.1DE00F	111FF.87A	
	FFFFFF.FFFFFF			

7. Efectuar las siguientes restas utilizando el complemento del sustraendo:

a) 111111-101111

b) 100001-100000

c) 1000000-110111

d) 10010011-1111101

2.5.2 Representación interna de la información

1. Sean los siguientes números expresados en base 10:

12 17 25 99 123 200 -12 +13 -100 155 -108 -234 -128

Expresarlos:

a) Binario en signo-magnitud, con una longitud de 8 dígitos binarios

b) Binario en complemento a 1, con una longitud de 9 dígitos binarios

c) Binario en complemento a 2, con una longitud de 8 dígitos binarios

2. Interpretar las siguientes combinaciones binarias:

1010 10001 0011 111001 1010000

a) Como datos binario natural

b) Como datos signo-magnitud

c) Como datos en complemento a uno

d) Como datos en complemento a dos

3. Codificar en formato IEEE 754 con s=1, ne=3, nm=4

13 -13 0.8 -8.25 -0.125 32.5

4. Suponiendo la representación en punto flotante del ejercicio anterior, interpretar los siguientes valores

a) 10100101

b) 01001110

c) 01101101

d) 10000000

5. Usando un editor de textos hemos creado y almacenado en el disco duro de nuestro ordenador un fichero de texto. A continuación, y mediante la herramienta adecuada, visualizamos su contenido sin decodificar, o sea, el conjunto de 0's y 1's en que ha sido traducido el conjunto de caracteres escrito, para poder ser almacenado y manipulado por el ordenador. Suponiendo la salida de la herramienta de visualización en hexadecimal, indicar cual será el texto que hemos escrito si lo que aparece es:

```
31 2C 32 2C 33 0D 45 53 54 4F 20 65 73 20 75 6E 61 20 70 72 75 65 62
61 2E
```

2.5.3 Álgebra de Boole

1. Simplificar $w = xy + \overline{yx}z$

2. Demostrar:

a) $a + ab = a$ b) $a + \bar{a}b = a + b$

c) $\overline{a + b} = \bar{a} \cdot \bar{b}$ d) $\overline{(\bar{a})} = a$

3. Un estudiante ve en un boletín unas condiciones para matricularse. Encontrar una expresión más sencilla.

1. Tener 60 créditos y estudiar ingeniería y buen expediente.
2. O bien tener 60 créditos y estudiar ingeniería y aceptación del departamento.
3. O bien menos de 60 créditos y estudiar ingeniería sin buen expediente.
4. O bien buen expediente y aceptación del departamento.
5. O bien estudiante de ingeniería y sin aceptación del departamento.

4. Diseñar una función que se comporte como un detector de mayoría: “Que dé un 1 cuando la mayoría de las entradas estén en 1”. Para 3 variables.

3. Sistemas operativos

3.1 Qué es un sistema operativo

Antes de pasar a estudiar un sistema operativo en particular, debemos contestar a la pregunta de ¿qué es un sistema operativo? y ¿para qué nos sirve? Podríamos dar varias definiciones posibles, sin embargo, vamos a ver el sistema operativo en sus dos aspectos más habituales: como gestor del ordenador y como mediador entre el usuario y el ordenador.

En primer lugar, debe quedar claro que un sistema operativo es un conjunto de programas, es decir, nos encontramos ante varios elementos software programados para un máquina específica. Estos programas se encargan de varias tareas.

En su papel de **gestor** del ordenador, el sistema operativo debe gestionar los recursos del sistema informático (procesadores, memoria, discos, etc), entre los diferentes procesos (programas en ejecución) que compiten por ellos. Esto lo podemos ver más claramente en un sistema multitarea donde las tareas de gestión son más complicadas: supongamos que tenemos un par de programas ejecutándose; en un navegador de Internet estamos cargando una página y como tarda mucho, aprovechamos para ejecutar un programa de cálculo científico. Dado que existen dos programas en ejecución simultánea y el microprocesador de un ordenador habitualmente es único y por él tienen que pasar todas las instrucciones, queda claro que "alguien" debe decidir cuál de los dos programas tiene acceso, en un momento dado, al microprocesador. Lo mismo podemos decir del resto de los recursos como memoria, discos, etc. Si pensamos en un sistema operativo multiusuario y suponemos que dos usuarios quieren imprimir simultáneamente en la misma impresora, el sistema operativo deberá decidir que uno de los usuarios puede imprimir en ese momento y que el trabajo del otro usuario quede almacenado en una cola de impresión para cuando la impresora quede libre.

En su papel de **mediador**, el sistema operativo ofrece al usuario que utiliza el ordenador una especie de "máquina virtual" o "máquina extendida" más fácil de utilizar que si tuviera que acceder directamente al hardware. Pongamos como ejemplo un comando como *copy* que permite copiar un archivo de una ubicación a otra dentro de un disco o entre diferentes discos. Para realizar estas operaciones de forma correcta, deberíamos ser capaces de encontrar el archivo dentro del disco, una vez encontrado posicionar la cabeza lectora del disco en el punto correcto y comenzar a transmitir la información a la memoria y de la memoria a la ubicación destino. Pues bien, todas estas tareas de bajo nivel y que requieren un conocimiento del funcionamiento hardware del ordenador

están ya programadas y nosotros, como usuarios del ordenador, no tenemos que preocuparnos, ya que el sistema operativo se encarga de ello; en el ejemplo que poníamos, el usuario sólo debe usar el comando *copy* dando como parámetros la ubicación origen y la ubicación destino. Este procedimiento por el cual el usuario "ve" una máquina virtual más fácil de utilizar que el hardware subyacente se conoce como "Principio de embellecimiento".

Como resumen, podemos definir el sistema operativo de la siguiente manera:

> El sistema operativo es un conjunto de programas cuyas misiones son:
>
> a) Gestionar los recursos del sistema informático (procesadores, memoria, discos, etc), entre los diferentes procesos que compiten por ellos, y
>
> b) Ofrecer al usuario una especie de "máquina virtual" o "máquina extendida", más fácil de usar que el hardware subyacente ("Principio de embellecimiento").

3.1.1 Clasificación de los sistemas operativos

Podemos realizar una clasificación de los diversos sistemas operativos que existen atendiendo a varios criterios o características de estos. Estos criterios pueden ser atendiendo a las tareas, a la planificación o a la gestión de memoria.

Tareas

En este punto clasificamos los sistemas operativos atendiendo al número de tareas que puede atender simultáneamente. Tenemos dos tipos:

- *Monotarea*: el sistema operativo solamente puede atender una tarea en un momento dado. Un ejemplo de S.O. de este tipo es MS-DOS.
- *Multitarea*: el sistema operativo puede atender varias tareas a la vez. A su vez estas tareas pueden provenir de un único usuario o de varios usuarios, lo cual dependerá de las capacidades del sistema operativo. Dentro de los sistemas operativos multitarea, existen los S.O. monousuario (por ejemplo, Windows NT) y multiusuario (por ejemplo, VMS y UNIX), donde el S.O. puede atender a un único usuario o a varios en la misma máquina, respectivamente.

Planificación

La planificación de un S.O. define cómo se reparte el tiempo de CPU entre los diversos procesos. Por supuesto, esto sólo tendrá sentido en S.O. multitarea

donde puede ocurrir que en un momento dado varios procesos quieran utilizar el microprocesador y como este es único, debe especificarse la política de acceso. Existen varias formas de realizar esta planificación:

- *Tiempo compartido* (Round-Robbin): se asigna el mismo tiempo para cada uno de los procesos.
- *Prioridades*: cada proceso tiene asignada una prioridad y hasta que no termina un proceso su ejecución, no se cede la CPU al siguiente. Estas prioridades pueden ser a su vez estáticas (fijas, no se modifican) o dinámicas (existen ciertos criterios para cambiarlas implementados en el S.O.).
- *Mixtas*: existe una planificación concreta a base de asignar tiempos en función de prioridades; en el caso de que dos procesos tengan asignada la misma prioridad, se comparte el tiempo entre los dos.

Lo habitual es tener planificación mixta, lo cual ocurre en los sistemas operativos VMS y UNIX. En estos casos, a aquellos procesos poco activos se les suele dar una prioridad máxima (por ejemplo, un editor de textos) y aquellos que exigen mucho tiempo de computación, una baja prioridad (por ejemplo, una inversión de matrices). Esto tiene sentido porque un proceso poco activo como un editor de textos consume muy pocos recursos de CPU, es decir, dentro de un intervalo de tiempo dado, el tiempo de CPU que va a usar es muy poco ya que desde que el usuario pulsa una tecla hasta que pulsa la siguiente, el ordenador ha tenido tiempo de realizar otras muchísimas tareas. Sin embargo, sería incómodo para el usuario que desde que pulsa una tecla hasta que aparece el resultado en la pantalla, pasara mucho tiempo; esa es la razón para darle máxima prioridad. En el otro extremo están los procesos que requieren mucho tiempo de computación, es decir, mucho tiempo de acceso a la CPU, como una inversión de matrices, una integral numérica, etc. Con el objeto de no saturar la máquina con estos cálculos, y debido a que habitualmente no existe una restricción temporal para acabar los cálculos, se le asigna una prioridad baja.

Gestión de memoria

Existen dos maneras básicas de gestionar la memoria:

- Memoria real
- Memoria virtual

El S.O. que sólo utiliza memoria real quiere decir que el único lugar donde le es posible cargar el código de un programa es en la memoria física real, es decir, en la RAM. En implementaciones de S.O. que utilizan memoria virtual, es posible hacer uso de espacio de almacenamiento en disco como si fuera

memoria adicional de la que dispone el ordenador, es decir, la memoria efectiva puede ser mayor que la real.

3.2 UNIX y MS-DOS

Vamos a estudiar brevemente dos de los sistemas operativos más extendidos hoy en día: UNIX y MS-DOS. Trataremos de aprovechar sus semejanzas para facilitar el estudio, ya que aunque se trata de dos sistemas operativos de naturaleza distinta en cuanto a su concepción, el tipo de tareas que se puede realizar con ellos es semejante.

Enlazando con el apartado anterior, podemos decir que UNIX es un sistema operativo:

◊ Multitarea,

◊ Multiusuario,

◊ Planificación mixta,

◊ Casi todas las implementaciones son de memoria virtual.

MS-DOS, por el contrario, es un sistema operativo:

◊ Monotarea,

◊ Monousuario,

◊ Memoria real.

Al realizar la migración de MS-DOS a MS-Windows, el S.O. accede a capacidades multitarea y de gestión de memoria virtual, sin embargo, estas capacidades resultan más pobres que en UNIX.

3.2.1 Breve historia de Unix

El S.O. Unix se gestó a finales de los años sesenta en los laboratorios Bell AT&T sobre un ordenador PDP-7. La razón de su origen se debe a que Ken Thompson, insatisfecho con el sistema operativo que utilizaba en su trabajo, decidió escribir su propio S.O. Inicialmente fue escrito en lenguaje ensamblador, pero más adelante se reescribió parte del sistema operativo en un nuevo lenguaje de programación denominado B (precursor del actual lenguaje C). Al mismo tiempo, otro programador de la misma compañía, Dennis Ritchie, padre del lenguaje C, entró en contacto con Unix y, junto con Ken Thompson, tradujo el Unix a este lenguaje.

Dada la imposibilidad de comercialización por parte de AT&T, se decidió distribuirlo con fines altruistas a Universidades, a cambio de un pago simbólico. Esta decisión tuvo dos consecuencias:

- Rápida extensión y uso en el mundo científico.

- Diversidad de versiones ya que no había quién dirigiera su desarrollo y evolución.

Para paliar este último inconveniente, en 1984 AT&T lanza el estándar Unix System V.

3.2.2 Breve historia de MS-DOS

Las iniciales DOS se refieren a *Disc Operating System* o *Sistema operativo de disco*. MS-DOS se refiere al desarrollado por Microsoft para ordenadores personales. Su gran popularidad en los ordenadores personales se debe a su adopción por IBM cuando aparece en escena el ordenador personal IBM-PC en 1981, aunque en ese entorno se le conocía como PC-DOS. Microsoft, responsable del MS-DOS, ha ido desarrollando nuevas versiones a medida que aumentaban las prestaciones de los PC y popularizando el uso de entornos gráficos con los sucesivos desarrollos de MS-Windows.

3.2.3 Diferencias y similitudes

Ambos sistemas operativos disponen de un **intérprete de comandos**, al que daremos órdenes y que serán ejecutadas por el ordenador. La primera diferencia entre ambos sistemas operativos la encontramos en el prompt de espera de órdenes.

3.2.4 Prompt

En Unix, el prompt puede tomar muchas formas ya que de hecho es configurable por el propio usuario. Habitualmente consiste en un signo de dólar ($) . Como se ha dicho, el usuario puede modificar el prompt, pero no solamente en lo que atañe al signo que aparecerá en pantalla sino que se puede modificar para que se incluya información adicional y eventualmente variable como el nombre del usuario, el nombre de la máquina, el directorio local, etc. En Unix existen prompts adicionales; habitualmente trabajaremos con el denominado prompt primario, pero si, por ejemplo, se invoca un intérprete de comandos adicional desde el actual, el prompt cambiaría al denominado prompt secundario que normalmente es el signo > , aunque al igual que el primario, es configurable por el usuario.

En DOS, el prompt es también configurable por el usuario y la forma habitual en que está dispuesto es: letra de unidad de disco + directorio actual + signo > . De esta manera, si nos encontramos en el directorio WINDOWS ubicado en la unidad de disco C, el prompt aparecería como C:\WINDOWS>

3.2.5 Jerarquía de ficheros

La jerarquía de ficheros puede parecer, a primera vista, muy parecida en ambos sistemas operativos, sin embargo, difieren significativamente.

Comencemos por las diferencias. En la jerarquía de archivos en Unix no tiene sentido hablar de unidad de disco local ya que desde el punto de vista del usuario, sus ficheros se encuentran contenidos en directorios, pero estos directorios no se sabe en que disco están; pueden encontrarse en un disco local de la máquina pero incluso pueden estar ubicados en otras máquinas remotas. En DOS, sí existe el concepto de disco dentro de la jerarquía de archivos y es preciso hacer referencia a él a la hora de ubicar la ruta jerárquica donde se encuentra un fichero en particular. Otra diferencia, aunque menos significativa, es el hecho de que el símbolo separador de directorios en Unix es el símbolo /, y en DOS es \. En cuanto a las analogías, en ambos sistemas operativos existe el concepto de directorio raíz que equivale a la cúspide del árbol de directorios del que se ramifican el resto; sin embargo, en Unix el directorio raíz es único dentro de la jerarquía de archivos, y en DOS existe un directorio raíz en cada uno de los discos.

3.2.6 Desplazamiento por la jerarquía de archivos

La forma de desplazarse por la jerarquía de archivos es semejante en ambos sistemas operativos. El directorio actual en el que nos encontramos se referencia como **.** y el directorio padre del que nos encuentramos como **..** (Figura 3-1). Para desplazarnos por los directorios, hacemos uso del comando `cd` (cambio de directorio) con un argumento que indica al directorio al que nos queremos desplazar. La forma de dar el argumento puede ser a su vez de dos tipos: acceso absoluto o acceso relativo. Todo esto lo explicamos más adelante.

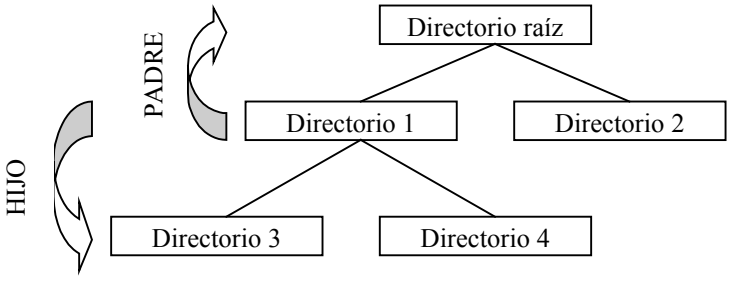


Figura 3-1. Terminología de directorio padre-hijo

3.3 Interfaces gráficas: Windows

Hace varios años era común que al trabajar en un ordenador con el sistema operativo que fuera (MS-DOS, Unix, VMS, etc.), encontrarnos con un terminal de sólo texto donde debíamos dar las órdenes a través del teclado. Hoy en día es habitual trabajar en el ordenador con interfaces gráficas donde aparecen una serie de iconos que al seleccionarlos arrancan determinados programas, y una serie de menús desplegables que nos permiten modificar

aspectos de configuración y ejecutar aplicaciones. Los entornos de ventanas se empezaron a popularizar con los ordenadores Apple, aunque en el despegue definitivo tuvo mucho que ver Microsoft con sus entornos Windows.

Para ordenadores personales, Microsoft ha ido desarrollando versiones cada vez más mejoradas de sus sistemas de ventanas, siendo los más conocidos, por orden de aparición, Windows 3.0, 3.1, 95, 98, *me*, y Windows NT y 2000 para ordenadores de alta gama.

En estaciones de trabajo que ejecutan Unix también ha habido un desarrollo paralelo de entornos gráficos que facilitan el uso del ordenador y lo hacen más agradable. En estos casos hablamos de sistemas X-Windows y aquí, en función del fabricante de la estación de trabajo, nos encontramos con diferentes versiones y denominaciones.

3.4 Estructura de ficheros / directorios

3.4.1 Unix

Vamos a comenzar explicando las características del sistema de ficheros que utiliza Unix y, posteriormente, veremos las diferencias que se introducen al utilizar MS-DOS.

UNIX emplea un sistema de ficheros jerárquico de directorios-ficheros (ver Figura 3-1).

No existe, a nivel de usuario, el concepto de *volumen*, ni de *dispositivo físico*. Es decir, el usuario no sabe en qué disco están los ficheros que está utilizando, como comentábamos en un punto anterior.

El elemento clave en esta jerarquía es el fichero (o archivo).

> Fichero (o archivo): "conjunto de información al que se le da un nombre"

Existen tres tipos de ficheros en UNIX:

- Ordinarios: Son cadenas de bytes terminadas con <ctrl>D (este código de control significa fin de fichero). Pueden contener texto, objetos, ejecutables, bibliotecas de módulos, etc.
- Directorios: Contienen nombres de ficheros y su dirección física. Puede pensarse en ellos como carpetas (de hecho esta es la denominación en entornos de ventanas) que contienen ficheros y directorios. Un directorio dentro de otro directorio se denomina subdirectorio.
- Especiales: Asociados a dispositivos entrada/salida. Contienen referencias a los drivers (programas que manejan directamente los dispositivos y que forman parte del núcleo). Pueden ser de tipo "bloque" (apuntan a

dispositivos tipo disco) y "carácter" (apuntan a dispositivos como terminales, impresoras, etc). Por convenio, residen en el directorio /dev.

Al elegir los *nombres de los ficheros*, es conveniente limitarse a utilizar sólo los caracteres que correspondan a letras, números, el carácter subrayado _ y el carácter punto `.`. Los ficheros cuyo nombre comience por punto permanecen ocultos.

En UNIX existe una jerarquía de directorios que para un sistema estándar sería:

/ Raíz

- /dev Ficheros especiales de dispositivos
- /lib Bibliotecas del sistema
- /bin Órdenes más empleadas
- /etc Datos y órdenes restringidas al superusuario
- /tmp Ficheros temporales (se borra periódicamente)
- /usr Órdenes, bibliotecas y programas adicionales
  - /usr/lib
  - /usr/bin
  - /usr/man
  - /usr/...
- /users Directorios de usuarios (puede aparecer también como /home)

Los ficheros se especifican por:

```
{camino jerárquico}/nombre{.ext}
```

donde las llaves (`{}`) indican que el contenido puede ser o no necesario. Los ficheros pueden constar de una extensión que es lo que aparece tras el punto (`.`). Un ejemplo de especificación de fichero sería:

```
/usr/lib/starbase/demos/star.f
```

Con esto estaríamos diciendo que queremos hacer referencia al fichero ordinario denominado `star.f` que se encuentra en el directorio `demos` que a su vez se encuentra en el directorio `starbase` que a su vez se encuentra en el directorio `lib` que a su vez se encuentra en el directorio `usr` que a su vez "cuelga" del directorio raíz. Dada la estructura jerárquica tipo árbol, al final, todo "cuelga" del directorio raíz.

El directorio raíz es el único que no tiene nombre (se hace referencia a él como /). Cada usuario tiene un directorio *HOME* que es el directorio asignado a ese

usuario para que almacene sus ficheros. El camino de este directorio está contenido en la variable HOME.

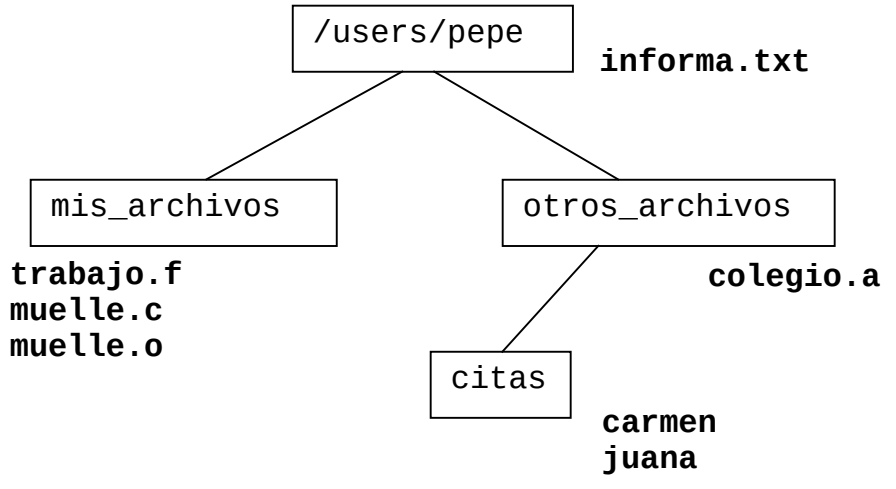
Los ficheros pueden tener cualquier longitud hasta 256 caracteres. Como se ha dicho anteriormente, la extensión es opcional. En caso de que exista, por convenio significa:

- .f Programa fuente escrito en lenguaje FORTRAN
- .p Programa fuente escrito en lenguaje PASCAL
- .c Programa fuente escrito en lenguaje C
- .o Fichero objeto
- .a Biblioteca de módulos
- .h Fichero de "cabecera"

Un fichero puede ser referenciado por su nombre precedido del nombre del directorio o directorios que le contiene de la manera que pasamos a explicar a continuación. En cualquier momento, los comandos que se teclean al intérprete de comandos están dados en referencia al directorio actual, es decir, el directorio en el que nos "encontramos". Cuando nos referimos a un fichero lo podemos hacer en referencia al directorio actual (vía de **acceso relativa**) o en referencia al directorio raíz (vía de **acceso absoluta**). En la vía de acceso relativa podemos hacer uso de la notación `..` que significa una referencia al directorio padre. Para clarificar todo esto, pongamos un ejemplo:

Ejemplo

Supongamos la siguiente estructura de ficheros-directorios:



y que nos encontramos situados (más adelante explicaremos cómo nos *situamos* en un determinado directorio) en el directorio `otros_archivos`. Las referencias a los diferentes ficheros se harían de la siguiente manera:

Nombre del fichero	Acceso absoluto	Acceso relativo
informa.txt	/users/pepe/informa.txt	../informa.txt
trabajo.f	/users/pepe/mis_archivos/trabajo.f	../mis_archivos/trabajo.f
muelle.c	/users/pepe/mis_archivos/muelle.c	../mis_archivos/muelle.c
muelle.o	/users/pepe/mis_archivos/muelle.o	../mis_archivos/muelle.o
colegio.a	/users/pepe/otros_archivos/colegio.a	colegio.a
carmen	/users/pepe/otros_archivos/citas/carmen	citas/carmen
juana	/users/pepe/otros_archivos/citas/juana	citas/juana

o

Normalmente, será más corto y sencillo utilizar el acceso relativo excepto en determinados casos (por ejemplo, en el ejemplo anterior, `/bin/fgrep`).

UNIX no mantiene versiones de ficheros, por lo que es necesario prestar especial atención a acciones como borrarlos o modificarlos.

Existen 3 ficheros estándar implementados en UNIX y que es importante conocer para realizar determinadas acciones:

- Entrada estándar (`stdin`): Teclado (0)
- Salida estándar (`stdout`): Pantalla (1)
- Errores estándar (`stderr`): Pantalla (2)

La redirección entrada/salida, que se explicará más adelante, permite cambiar estas asignaciones en cualquier momento.

3.4.2 MS-DOS

MS-DOS utiliza también un sistema jerárquico de directorios-ficheros, pero a diferencia de UNIX, sí existe para el usuario el concepto de *volumen* o *dispositivo físico*, más conocido por **unidad de disco**. Estas unidades de disco llevan asociadas una **letra de unidad** que debe aparecer en todos los accesos absolutos que hagamos a los ficheros. Tradicionalmente se utilizan las siguientes asignaciones:

- A: unidad de disco flexible de 3½''
- B: unidad de disco flexible de 5¼''
- C: unidad de disco duro
- D: unidad de CD-ROM

aunque en función del ordenador podría cambiar. Por ejemplo, en los ordenadores actuales ya no se utilizan unidades de disco flexible de 5¼'' por lo que la letra de unidad B no se utiliza o se reasigna a la A. En determinados

casos, nos encontramos con ordenadores con varios discos duros o discos duros "particionados" por lo que a efectos prácticos nos encontramos con que varias letras de unidad consecutivas desde la C hacen referencia a distintas unidades de disco duro.

Los conceptos de ficheros ordinarios y directorios se mantienen en MS-DOS, aunque no existen los ficheros especiales. Es decir, todas las referencias a dispositivos se solucionan a través de puertos: LPT para puertos paralelos, COM para puertos serie, etc.

Se pueden hacer las mismas recomendaciones que hacíamos en Unix a la hora de elegir los nombres de los ficheros. En MS-DOS, sin embargo, la forma de ocultar un fichero no se realiza haciendo que comience por punto, sino que se consigue modificando las propiedades del fichero.

En MS-DOS no existe una jerarquía de directorios del sistema tan normalizada ni tan estructurada como en Unix. Cuando el usuario arranca el ordenador, se encuentra con que en el directorio raíz de la unidad C aparecen los archivos principales del sistema, y en el directorio DOS aparecen los comandos típicos del sistema operativo.

Como decíamos anteriormente, la especificación de los ficheros se realiza de forma similar a Unix, salvo que debemos preceder el camino jerárquico por la letra de la unidad y cambiar los / por \.

```
{letra de unidad}:{camino jerárquico}\nombre{.ext}
```

Un ejemplo sería:

```
C:\USER\STAR.FOR
```

Con esto estaríamos diciendo que queremos hacer referencia al fichero ordinario denominado STAR.FOR que se encuentra en el directorio USER que a su vez se encuentra en el directorio raíz de la unidad de disco C.

Como MS-DOS es un sistema operativo monousuario, no se suele hacer distinciones de directorios por usuarios.

En cuanto a la longitud de los nombres de archivos, existe una restricción más fuerte que en Unix ya que el nombre debe tener una longitud máxima de 8 caracteres y la extensión, que siempre existe aunque pudiera estar vacía, tiene una longitud máxima de 3 caracteres. Es decir, que el nombre debe tener la forma ????????.??? donde el signo de interrogación puede ser cualquier carácter excepto los especiales que veremos más adelante. Esta limitación de longitud desaparece en las últimas versiones del sistema operativo que se distribuyen ya con el interface gráfico a partir de Windows 95; sin embargo, al permitir que el carácter *espacio* forme parte del nombre, a menudo es necesario encerrar el nombre entre comillas para que no dé lugar a confusión. MS-DOS también asume que la extensión indica el tipo de archivo; tomando los mismos ejemplos que vimos en Unix:

- .FOR Programa fuente escrito en lenguaje FORTRAN
- .PAS Programa fuente escrito en lenguaje PASCAL
- .C Programa fuente escrito en lenguaje C
- .OBJ Fichero objeto
- .LIB Biblioteca de módulos
- .H Fichero de "cabecera"

Las referencias por acceso absoluto o relativo funcionan de manera semejante a Unix, teniendo en cuenta lo que ya hemos comentado previamente: necesidad de letra de unidad en accesos absolutos y uso del símbolo \ como separador de directorios. Supongamos el mismo ejemplo que vimos en la página 69 y veamos cómo haríamos referencia a los diversos ficheros desde MS-DOS y asumiendo que nos "encontramos" en el directorios `otros_archivos`:

Nombre del fichero	Acceso absoluto	Acceso relativo
informa.txt	c:\users\pepe\informa.txt	..\informa.txt
trabajo.f	c:\users\pepe\mis_archivos\trabajo.f	..\mis_archivos\trabajo.f
muelle.c	c:\users\pepe\mis_archivos\muelle.c	..\mis_archivos\muelle.c
muelle.o	c:\users\pepe\mis_archivos\muelle.o	..\mis_archivos\muelle.o
colegio.a	c:\users\pepe\otros_archivos\colegio.a	colegio.a
carmen	c:\users\pepe\otros_archivos\citas\carmen	citas\carmen
juana	c:\users\pepe\otros_archivos\citas\juana	citas\juana

Como se puede comprobar, la única diferencia es la apuntada previamente.

En el ejemplo que hemos visto, hemos cometido deliveradamente un error; ¿sabes cúal es? Efectivamente, estamos usando *otros_archivos* como nombre de directorio y en MS-DOS hemos dicho que el límite máximo en la longitud de un archivo son 8 carácteres, por lo que *otros_archivos*, al tener longitud 14, no estaría permitido; deberíamos usar otro nombre como por ejemplo, *OTROS_AR*.

En MS-DOS existe la posibilidad, en ciertos casos, de recuperar un fichero borrado a través del comando *undelete*. Esto es así debido a que cuando se borra un archivo, no se borra su contenido en el disco, sino solamente la referencia que une el nombre con su contenido. El espacio en disco que ocupaba pasa a estar "disponible" para guardar más información, por lo que si este espacio no ha sido ocupado, la recuperación es posible.

3.4.3 Juegos de caracteres

UNIX

El conjunto de caracteres que podemos usar en Unix se puede clasificar en los siguientes grupos:

- Letras minúsculas: `a,b,c,...,z`
- Letras mayúsculas: `A,B,C,...,Z`
- Números: `0,1,...,9`
- Caracteres Especiales: `\ / - _ . ; [ ] ( ) < > & |` y los metacaracteres

Es de destacar que en Unix, la misma letra en minúscula y en mayúscula representa distinto carácter por lo que sería diferente un archivo llamado *lista* que un archivo llamado *Lista*. Los comandos, en particular, suelen estar en minúscula.

Los caracteres especiales no deben formar parte del nombre (en determinadas condiciones, se puede hacer pero no es recomendable) ya que tienen un significado especial para el intérprete de comandos. En particular, los metacaracteres hacen referencia a los símbolos asterisco ***** e interrogación **?**. El asterisco representa cualquier cadena de caracteres y la interrogación representa cualquier carácter. Se suelen utilizar para hacer referencia a varios archivos con un sólo comando, por ejemplo, si queremos borrar todos los ficheros fuente en pascal que empiezan por la letra *b*, en vez de borrar uno a uno, diríamos que queremos borrar el archivo *b*.p*. De la misma manera, si queremos borrar aquellos cuya segunda letra sea una *b*, usaríamos el nombre *?b*.p*.

El resto de caracteres especiales tienen el siguiente significado:

`;`	Equivale a un retorno de carro en la línea de órdenes.
`[]`	Engloba grupos de caracteres, separados por coma `,` o por guión `-`. Por ejemplo, `fich[1,2].f` se refiere tanto a `fich1.f` como a `fich2.f`.
`<>`	Sirven para redireccionar entrada / salida, respectivamente (se explica más adelante, en el apartado 3.6.3).
`&`	Envía un proceso en "background" (se ejecuta como subproceso del intérprete de comandos).
`|`	Construcción de tuberías o "pipes" (apartado 3.6.3).

\ Suprime el significado especial del carácter que le sigue. Por ejemplo, * dejaría de ser equivalente a cualquier cadena de caracteres.

MS-DOS

El juego de caracteres a usar en MS-DOS es el mismo que en Unix, aunque en este caso, el sistema operativo no es sensible a mayúsculas/minúsculas, es decir, es equivalente escribir *lista* que *Lista* que *LISTA*. De hecho, habitualmente se usan las mayúsculas. En cuanto a los caracteres especiales, el asterisco y la interrogación tienen el mismo significado que en Unix. También los símbolos < y > permiten redireccionar la entrada y la salida así como el símbolo | sirve para construir tuberías.

3.5 Clasificación de usuarios en Unix

UNIX es un sistema operativo multitarea y multiusuario, por lo que se deben establecer ciertos mecanismos de tal manera que, simultáneamente, se protejan los datos de un usuario frente a otros y éstos puedan ser compartidos en caso necesario. UNIX posee un mecanismo de *permisos* asociados a cada fichero. Este mecanismo permite que los ficheros y directorios pertenezcan a un usuario en particular. UNIX también permite que los ficheros sean compartidos entre usuarios y grupos de usuarios. El comportamiento por defecto en la mayoría de los sistemas es que todos los usuarios pueden leer los ficheros de otro usuario, pero no pueden modificarlos o borrarlos.

Los grupos de usuarios se definen normalmente en función del tipo de usuario. Por ejemplo, en una Universidad, los usuarios pueden clasificarse como estudiantes, profesores, invitados, etc.

Cada usuario (perteneciente a un grupo de usuarios) tiene asociado un nombre, una palabra clave o password, un directorio y un proceso de arranque:

- Nombre: Identificación del usuario cuando entra en la máquina (*login*).
- Clave: Palabra oculta que sólo conoce el usuario y que le permite, junto con el *login*, acceder al sistema.
- UID, GID: Números de identificación de usuario y grupo, respectivamente.
- Directorio: Directorio inicial donde se situará el usuario al entrar en el sistema.
- Proceso: Primer proceso que se arranca una vez dentro del sistema.

Existen diferentes categorías de usuarios en función de sus privilegios (lo que puede y no puede hacer):

- Superusuario o root: Es el administrador del sistema. Tiene todos los privilegios.
- Usuarios normales: El resto de usuarios que pertenecen a distintos grupos, los cuales pueden tener una serie de propiedades comunes.
- Usuarios especiales: Asignados a tareas específicas por el sistema, generalmente de información o manejo de aplicaciones ya instaladas de uso común a usuarios externos o internos. Por ejemplo: mail (se encarga de recoger el correo y repartirlo a los diversos usuarios), lp (se encarga de aceptar trabajos de impresión y mandarlos a la impresora), bin, admin, ...

Desde el punto de vista de un usuario, el carácter u significa el propio usuario, g significa el conjunto de usuarios que pertenecen a su mismo grupo, o significa el resto de usuarios. Estos caracteres serán reconocidos por ciertos comandos u órdenes.

Los permisos que llevan asociados todos los ficheros y directorios se clasifican en *lectura* (read, r), *escritura* (write, w) y *ejecución* (execute, x). Estos permisos se pueden asignar al propio *usuario* (u), al *grupo* (g) y al *resto* (o).

El permiso de lectura permite a un usuario leer el contenido del fichero, o, en el caso de directorios, obtener un listado de su contenido.

El permiso de escritura permite a un usuario escribir y modificar el fichero. Para directorios, permite al usuario crear nuevos ficheros dentro del directorio o borrar los que contiene.

El permiso de ejecución permite a un usuario ejecutar un fichero (debería ser un programa o script —fichero que contiene órdenes para el sistema). En el caso de directorios, permiso de ejecución significa que el usuario puede introducirse en dicho directorio.

Debido a todos estos mecanismos, necesarios en un sistema operativo multiusuario, el trabajo sobre él implica una secuencia de entrada para acceder al sistema y, lo que es más importante, **recordar salir del sistema** con el comando **exit** para que nadie pueda trabajar con nuestros privilegios.

3.6 Comandos

Tanto en Unix como en MS-DOS, el formato general de los comandos es:

```
verbo     argumento argumento ...
```

donde el *verbo*, o nombre del comando, es siempre necesario y los *argumentos* pueden ser necesarios o no, dependiendo del comando u orden. Los argumentos pueden ser de dos tipos: opciones o adverbios, y nombres de ficheros. Las opciones van precedidas en Unix por un guión -, y en MS-DOS por el símbolo /.

A continuación, pasamos a describir los comandos más habituales con las opciones, en caso de que existan, más comunes. Indicaremos simultáneamente la forma en Unix en minúsculas y la forma en MS-DOS en mayúsculas, así como las opciones de Unix precedidas de un guión y las de MS-DOS precedidas del símbolo de dividir.

3.6.1 Comandos de información general

date DATE

En Unix, muestra la fecha y hora del sistema. En MS-DOS, solamente la fecha y da la posibilidad de introducir una nueva.

who

Información de quién está en el sistema.

whoami

Información de quién está en este terminal. Aparecerá mi nombre de usuario (¿quién soy yo?).

pwd CD

Directorio en el que se está trabajando.

ps

Información de qué es lo que está haciendo el sistema. Si no le añadimos opciones, solamente dará información sobre mi terminal. Existen multitud de opciones, algunas de las más comunes son:

`ps -e` Información de todo el sistema

`ps -l` Formato largo

`ps -f` Información de los comandos que el sistema está procesando

Según las opciones que le suministremos al comando, aparecerán diversos campos, entre los que cabe destacar:

UID: Número de identificación del usuario

PID: Número de identificación del proceso

PPID: Número de identificación del proceso padre (aquel proceso que creó a éste)

PRI: Prioridad (cuanto más alto sea este número, menor es la prioridad)

NI: Número "nice" (es la prioridad efectiva; sólo se puede subir o lo que es lo mismo, bajar la prioridad)

TIME: Tiempo de CPU en *min:sg*

COMD: Nombre del comando que se está ejecutando

Un ejemplo típico sería:

```
$ps
PID TT STAT  TIME COMMAND
 24  3 S     0:03 (bash)
161  3 R     0:00 ps
```

man HELP

Permite obtener ayuda sobre los diversos comandos. Lleva como argumento el nombre del comando del que queremos obtener información.

more MORE

Comando que permite presentar en pantalla el contenido de un fichero de texto de una forma filtrada.

Se puede utilizar de dos formas diferentes:

1. `more fichero1` (en MS-DOS hay que redireccionar: `more<fichero1`)
2. `comando | more`

Presenta una pantalla (25 líneas) del fichero, y se queda esperando una orden, con un mensaje en la última fila:

`-more (? %)` en Unix o `-- Más --` en MS-DOS

Ante este mensaje se pueden tomar varias opciones. En Unix, en función de la tecla que se pulse, se ejecutan las siguientes acciones:

<espacio>	Siguiente pantalla.
<intro>	Siguiente línea.
/cadena	Búsqueda de cadena en el texto.
n	Siguiente aparición de la cadena.
:n	Siguiente fichero (en caso de que sean varios a visualizar).
v	Edita con vi el fichero en curso. Al terminar, se retorna a more.
q	Fin.

En MS-DOS, sin embargo, cualquier tecla que pulsemos provoca un salto hacia la siguiente pantalla, excepto si pulsamos <ctrl-C> lo que provoca la finalización del comando (esto último también es cierto en Unix).

3.6.2 Comandos de manipulación de ficheros

cat TYPE

Dirige el contenido de ficheros a la salida estándar (normalmente, la pantalla). Lleva como argumento el nombre del fichero.

cp COPY

Realiza copia de ficheros. Lleva dos argumentos, el fichero fuente (aquel que queremos copiar) y el fichero destino (donde lo queremos copiar). Si el destino es un archivo ordinario, lo que estamos haciendo es duplicar la información del fichero fuente con otro nombre. En caso de que el destino sea un directorio, estamos haciendo una copia en dicho directorio pero el nombre permanece el mismo.

mv MOVE

Traslado o cambio de nombre de un fichero. Este comando funciona exactamente igual que *cp* o *COPY* pero además desaparece el fichero fuente.

REN ó RENAME

Como MOVE, es decir, cambio de nombre de un fichero, pero dicho fichero no puede cambiar de ubicación.

rm DEL

Borra ficheros. Algunas de las opciones más importantes son:

`-i` `/P` Pregunta antes de borrar.

`-r` Borra un directorio de forma recursiva, borrando primero los ficheros que están en él.

ls DIR

Lista el contenido de un directorio. Las opciones más importantes son:

`-a` /A:H Lista todos los ficheros (incluso los ocultos).

`-t` /O:D Ordena por fecha de creación.

`-p` Marca los directorios.

`-r` Invierte el orden.

`-s` Indica el tamaño.

`-l` Formato largo. Indica protecciones, propietario, grupo, tamaño y fecha de creación de cada fichero.

`-F` Distingue entre ficheros ordinarios, directorios y ejecutables.

`/P` Se detiene después de llenar la pantalla de información.

`/W` Utiliza el formato de listado extendido.

`/S` Muestra los archivos del directorio y sus subdirectorios.

En Unix, a la hora de interpretar una lista de ficheros en formato largo, conviene recordar lo que se dijo sobre las **protecciones** o permisos. Recuérdese que existían 3 niveles de protección (lectura r, escritura w, ejecución x) sobre 3 tipos de usuarios (propietario u, grupo g, resto o). Al ejecutar la orden `ls -l`, se nos facilita la información sobre las protecciones del fichero con el formato:

```
-rwxrwxrwx
```

El primer carácter indica el tipo de fichero:

`-` Fichero ordinario.

`d` Directorio.

`b` Fichero especial tipo bloque.

`c` Fichero especial tipo carácter.

`l` Enlace o *link*.

Los otros nueve caracteres expresan, en grupos de 3 (para u, g, o), los permisos de acceso a ese fichero. Un guión `-` en un campo implica que no existe ese tipo de permiso. Un ejemplo clarificador sería:

`-rwxr-x---` El propietario puede leer, escribir y ejecutar. El grupo puede leer y ejecutar. El resto de usuarios no tiene ningún acceso.

En MS-DOS no tiene sentido hablar de permisos y de usuarios, pero cuando se ejecuta un DIR sin parámetros, por defecto nos sale información, por cada archivo, de la fecha y hora de modificación, si es un directorio aparacerá la palabra <DIR>, el tamaño y el nombre de archivo.

cd CD

Cambio de directorio. Si en Unix se ejecuta sin argumentos, el intérprete de comandos nos cambia al directorio principal del usuario (HOME) mientras que en MS-DOS sería equivalente al comando *pwd* de Unix.

Este comando lleva como argumento el directorio al que queremos cambiar, que puede estar referenciado como acceso absoluto o como acceso relativo. En ambos sistemas operativos, si el nombre es **..**, cambiamos al directorio padre del actual.

Un caso especial en MS-DOS se da cuando queremos cambiar de letra de unidad. En ese caso, en vez de usar el comando CD, debemos escribir simplemente la letra de la unidad de disco a la que queremos cambiar seguido del símbolo dos puntos **:**.

mkdir MKDIR

Crea un directorio. Lleva como argumento el nombre del directorio que queremos crear pudiendo estar referenciado como acceso absoluto o como relativo.

rmdir RMDIR

Borra un directorio. Para poder ejecutarlo con éxito, dicho directorio debe estar vacío.

chown

Cambia el propietario de un fichero (deberá ser nuestro para poderlo hacer). La sintaxis es `chown propietario fichero`

chgrp

Cambia el grupo de un fichero.

chmod

Cambia los permisos de acceso de un fichero. Actúa sobre el propietario (`u`), el grupo (`g`), o el resto (`o`), añadiendo (`+`) o quitando (`-`) los permisos (`rwx`). Por ejemplo:

`chmod o+r fichero`	Se da permiso de lectura al resto de usuarios.
`chmod g+wx fichero`	Se da permiso de escritura y ejecución al grupo.
`chmod o-rwx fichero`	Se quitan todos los permisos al resto de usuarios.

Existe la posibilidad de dar protecciones en *octal*.

Permisos en binario			Octal	Significado
u	g	o		
xxx	xxx	xxx		
000	000	000	000	No se puede hacer nada
001	000	000	100	El propietario puede ejecutar
111	000	000	700	
111	111	111	777	Totalmente desprotegido

Por ejemplo, con la orden `chmod 653 fichero`, estaríamos asignando permiso de lectura y escritura para nosotros, lectura y ejecución para los de mi grupo, y escritura y ejecución para el resto de usuarios.

passwd

Permite cambiar la palabra clave o *password* de entrada al sistema. Pide el antiguo y el nuevo dos veces, como confirmación. Es habitual que existan ciertas reglas de validez para las posibles contraseñas que se pueden elegir, como por ejemplo, que no coincida con el nombre del usuario, que tenga una longitud mínima, etc.

3.6.3 Redirección de entrada/salida

La ejecución de un comando típico en Unix responde al esquema de la Figura 3-2.

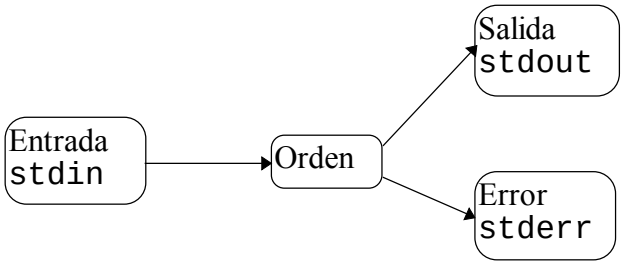


Figura 3-2. Ejecución de un comando u orden

En dicha figura se observa que el comando, si necesita algún dato de entrada, va a esperar que este se le introduzca a través de teclado, que es la entrada por defecto en el sistema operativo. Si el comando ha de devolver alguna

información respecto al resultado conseguido por su ejecución, por defecto lo enviará al dispositivo de salida por defecto que es la pantalla. De la misma manera, posibles errores en la ejecución del comando saldrían por el dispositivo de errores por defecto, que también es la pantalla.

Este comportamiento por defecto de los comandos (entrada por teclado, salida por pantalla) puede ser modificado a través de la denominada **redirección**.

Redirección simple

En la redirección simple, puede variarse la entrada/salida estándar, redirigiéndola de/a un fichero, utilizando el formato:

```
comando <entrada >[>]salida 2>[>]errores
```

(los corchetes indican que lo que contienen es opcional)

donde:

- `entrada` fichero que sustituye a `stdin`
- `salida` fichero que sustituye a `stdout`
- `errores` fichero que sustituye a `stderr`

La utilización de `>>` en lugar de `>` significa añadir al final del fichero, si ya existe, en vez de al principio, sustituyéndolo.

Los ficheros destino pueden ser alguno de los estándar. En este caso, se especifican con los símbolos `&1` (stdout), `&2` (stderr). Por ejemplo `cat <fichero1 >>fichero2 2>&1` redirecciona los errores a la pantalla.

Los fichero de salida y entrada deben ser diferentes.

Como ejemplo, para enviar un listado del contenido de un directorio a un archivo, lo realizaríamos de la siguiente manera:

`ls >listado.lst` (en UNIX)

`dir >listado.lst` (en MS-DOS)

Redirección encadenada

Permite que la salida de un comando se utilice como entrada al siguiente.

Formato:

`cmd1 | cmd2 | ..... | cmdN` "Pipeline" o "tubería"

La salida estándar de un comando de la "tubería" actúa como entrada estándar del siguiente.

Evidentemente, no tiene sentido redireccionar la entrada, salvo en el primer comando. Tampoco tiene sentido redireccionar la salida, salvo en el último comando.

Ejemplo:

`ls -l | more` (en UNIX)

`dir | more` (en MS-DOS, aunque el mismo efecto se consigue con `dir /p`)

3.7 Edición de texto

Un editor de texto es simplemente un programa que se usa para editar ficheros que contienen texto, tales como una letra, un programa en C, un fichero de configuración del sistema, etc. Los editores de texto son unas de las aplicaciones más necesarias y útiles en un sistema operativo, de ahí que habitualmente se incluye con este un editor más o menos avanzado.

A la hora de crear o modificar un archivo, se puede hacer uso tanto del editor de textos que se incluye normalmente con el S.O., o bien, se puede instalar y utilizar otro editor que nos guste más. Sin embargo, es conveniente estar familiarizado con el editor que incluye el S.O. ya que puede suceder que al acceder a un ordenador distinto al nuestro, nuestra única opción sea utilizar el editor por defecto.

El editor de texto por antonomasia del S.O. **Unix** es el denominado **vi** (*se debe leer uve-i*). Es un editor bastante primitivo orientado a texto y que al principio puede resultar complicado de utilizar, sin embargo, es un editor también muy potente que permite realizar acciones comunes de edición de manera muy rápida.

El editor de texto que viene incluido con **MS-DOS** se denomina **edit** y aunque es más fácil de utilizar que el vi, debido fundamentalmente a que incluye una barra de menú con las acciones que se pueden realizar, es sin embargo menos potente.

3.7.1 Editor vi de Unix

Las características típicas de este editor (**vi**sual editor) son:

- Observación inmediata de la modificación.
- Entorno edición-ejecución potente.
- Comandos invisibles.
- División del campo visual: trabajo y comandos.

A continuación vamos a dar una breve descripción de vi. No se discutirán todas sus características, solamente aquellas que creemos importantes para empezar. Para sacar todo el partido al editor, puede consultarse la ayuda de Unix, un manual de vi, o simplemente una tabla con los comandos más usuales.

La principal característica que puede sorprender a usuarios no habituados a este editor es que, en cualquier momento, se está en uno de tres modos de operación. Estos modos son conocidos como modo de comandos, modo de inserción, y modo de última línea.

Cuando se arranca vi, se está en el *modo de comandos*. Este modo permite usar ciertos comandos para editar ficheros o cambiar a otros modos. Por ejemplo, tecleando `x` en el modo de comandos, se borra el carácter bajo el cursor. Las teclas de dirección mueven el cursor por el fichero que se está editando. Generalmente, los comandos usados en el modo de comandos son de uno o dos caracteres de largo.

Insertar o editar texto se realiza en el *modo de inserción*. En vi, la mayor parte del tiempo se estará en este modo. Se puede iniciar el modo de inserción usando un comando como `i` (insertar) desde el modo de comandos. En el modo de inserción, se inserta texto en el documento desde la posición actual del cursor. Para finalizar el modo de inserción y volver al modo de comandos, se pulsa <ESC>.

El *modo de última línea* es un modo especial usado para ciertos comandos extendidos. Cuando se teclean estos comandos, aparecen en la última línea de la pantalla (de ahí su nombre). Por ejemplo, cuando se teclea `:` desde el modo de comandos, se salta al modo de última línea y se pueden usar comandos como `wq` (grabar el fichero y finalizar vi), o `q!` (finalizar vi sin grabar los cambios). El modo de última línea generalmente se usa para comandos que son más largos que un carácter. En el modo de última línea, se introduce un comando y se presiona <INTRO> para ejecutarlo.

3.7.2 Editor edit de MS-DOS

En este editor, al contrario que el anterior, siempre se está en disposición de escribir texto y de borrarlo utilizando las teclas específicas de borrado (DEL y SUPR). Para realizar acciones más complejas sobre el texto, aparece una serie de menús en la barra superior:

- Archivo (<alt-A>): Permite realizar opciones sobre el archivo como abrir, guardar, imprimir y salir de la aplicación.
- Edición (<alt-E>): Permite realizar opciones de edición sobre texto seleccionado como cortar, copiar, pegar.
- Búsqueda (<alt-B>): Permite realizar búsquedas en el texto.

- Opciones (<alt-O>)
- Ayuda (<alt-Y> o F1)

Como se ha dicho, este editor es muy sencillo de usar y es preferible disponer de ratón ya que muchas acciones se facilitan bastante.

3.8 Formatos de ficheros

En el trabajo habitual con el ordenador, prácticamente toda la información que maneja un usuario cualquiera está en forma de ficheros. Simplificando, el usuario se encuentra con dos tipos de ficheros: los ficheros ejecutables que le permiten arrancar programas y los ficheros que contienen datos asociados con un determinado programa. En este apartado vamos a ver los tipos de ficheros más comunes con los que se va a encontrar un usuario y qué tipo de información debe esperar encontrar.

Desde un punto de vista de contenido de los ficheros, podemos distinguir los siguientes tipos de ficheros:

Tipo de fichero	Contenido
Fichero por lotes	Contiene comandos del sistema operativo
Fichero binario	Contiene datos o instrucciones en formato binario
Fichero de comandos	Contiene comandos del sistema operativo
Fichero de datos	Contiene datos
Fichero directorio	Contiene marcas sobre ficheros que pertenecen a él
Fichero ejecutable	Contiene un programa o comandos en formato ejecutable
Fichero de librería	Contiene funciones en formato objeto
Fichero de mapa	Contiene un mapa de un programa
Fichero objeto	Contiene código compilado
Fichero de texto	Contiene texto (leíble) en formato ASCII

3.8.1 Ejecutables

Respecto a los ficheros ejecutables, la forma de distinguirlos es distinta en Unix que en MS-DOS. A su vez, en ambos sistemas operativos, un fichero ejecutable puede ser tanto un *script* o fichero de comandos que contiene órdenes al sistema operativo como un programa ejecutable propiamente dicho que el sistema operativo carga en memoria como código binario. Comencemos por Unix. Un fichero ejecutable en Unix no suele contener una extensión

particular, aun cuando determinados *scripts* llevan la extensión *.sh*, mas la única forma de distinguirlos es porque llevan el permiso de ejecución activado. En MS-DOS, donde las extensiones tienen mayor importancia, los *scripts* llevan extensión *.bat* y los programas ejecutables binarios llevan extensión *.exe*. Resumiendo:

Unix

- Permiso de ejecución
- Extensión **.sh** en scripts, aunque no es necesario

MS-DOS

- **.EXE** Ejecutable binario
- **.COM** Fichero de comandos
- **.BAT** *Script* o fichero con órdenes del sistema operativo

3.8.2 Formatos de texto ASCII

En este apartado vamos a enumerar los formatos más comunes cuyo contenido es texto ASCII estándar. Todos los formatos que vamos a ver a continuación pueden ser tratados (es decir, creados, modificados y visualizados) con un editor de textos estándar como los que vimos en la sección 3.7. Escribiremos la extensión correspondiente tanto en Unix (en minúscula) como en MS-DOS (en mayúscula).

- .txt, .TXT Texto de tipo general
- .c, .C Código fuente de un programa en lenguaje C
- .p, .PAS Código fuente de un programa en lenguaje PASCAL
- .f, .FOR Código fuente de un programa en lenguaje FORTRAN
- .BAS Código fuente de un programa en lenguaje BASIC

3.8.3 Formatos de ofimática

A continuación enumeraremos los formatos más habituales en entornos de ofimática, es decir, ficheros de datos que contienen información de un procesador de textos, de una hoja de cálculo, de una base de datos, de una presentación visual, etc. Todos estos formatos son propietarios de las compañías correspondientes y para su tratamiento es necesario disponer de los programas correspondientes.

- .DOC Documento del procesador de textos Microsoft Word
- .WP Documento del procesador de textos WordPerfect

- .tex, .TEX — Procesador de macros con control sobre el formato. Latex está basado en TeX
- .XLS — Documento de la hoja de cálculo Microsoft Excel
- .PPT — Documento del programa de presentaciones Microsoft PowerPoint
- .MDB — Documento del gestor de base de datos Microsoft Access

3.8.4 Formatos gráficos

En cuanto a ficheros que almacenan información de tipo gráfico, existen multitud de estándares y multitud de programas que permiten su visualización, tratamiento y conversión. Estos formatos pueden ser de dos tipos: mapa de bits o vectoriales. A su vez pueden incluir compresión de datos o no, y, en el caso de incluir compresión de datos, esta compresión puede ser con pérdidas o sin ellas, respecto al gráfico original. También es importante el hecho de que algunos formatos admiten mayor número de colores que otros y varias resoluciones. Entre los formatos más conocidos están los siguientes.

- .gif, .GIF — **G**raphics **I**nterchange **F**ormat. Mapa de bits y compresión de datos
- .jpg, .JPG — **J**oint **P**hotographic **E**xperts **G**roup. Compresión con pérdidas
- .BMP — **B**it **M**apped **G**raphics. Estándar de MS-Windows. Se almacena en un formato denominado *mapa de bits independiente de dispositivo* (DIB)
- .PCX — Desarrollado inicialmente por ZSOFT para el programa de PC Paintbrush. Mapa de bits
- .eps, .EPS — **E**ncapsulated **P**ost**S**cript. Usado por el lenguaje Postscript
- .tiff, .TIF — **T**agged **I**mage **F**ile **F**ormat.
- .WPG — Formato gráfico utilizado por el procesador de texto WordPerfect
- .WMF — **W**indows **M**etafile **F**ormat. Formato usado para intercambiar gráficos entre aplicaciones MS-Windows

3.8.5 Formatos de documentación

En este apartado incluimos aquellos formatos de ficheros que permiten la representación de información escrita para su distribución como

documentación en manuales, presentación de productos, artículos científicos, etc. Algunos de ellos almacenan la información en formato ASCII como en el caso del postscript y HTML, mientras que otros lo hacen en formato binario como en el caso del PDF.

- .ps, .PS — Fichero en lenguaje de descripción de páginas PostScript desarrollado por Adobe Systems. Orientado principalmente a la impresión de documentos en impresoras láser
- .html, .HTM — **H**yper**T**ext **M**arkup **L**anguage. Lenguaje usado para crear documentos en la World Wide Web (ver capítulo siguiente **Redes e Internet**)
- .PDF — **P**ortable **D**ocument **F**ormat. Desarrollado por Adobe Systems, para su visualización se precisa el programa Adobe Acrobat Reader

3.8.6 Formatos de sonido y vídeo

En este último apartado, veremos los formatos más comunes hoy día para almacenar sonido y vídeo. Este es un campo donde se está en constante evolución apareciendo nuevos estándares continuamente, por lo que es posible que dentro de unos años los estándares en sonido/vídeo sean otros distintos de los que aparecen a continuación.

- .WAV — Estándar de sonido en PC desarrollado por Microsoft e IBM
- .mp3, .MP3 — Extensión de MPEG para archivos de sonido
- .ra, .RA — **R**eal**A**udio. Estándar de sonido para su transmisión en tiempo real a través de Internet, desarrollado por RealNetworks
- .rm, .RM — Estándar de video para su transmisión en tiempo real a través de Internet, desarrollado por RealNetworks
- .MOV — QuickTime movie. Sistema de vídeo y animación desarrollado por Apple Computer
- .mpeg, .MPG — **M**oving **P**icture **E**xperts **G**roup. Formato de vídeo, compresión con pérdidas, se basa en almacenar solamente la información que cambia de un fotograma al siguiente
- .AVI — **A**udio **V**ideo **I**nterleave. Formato del estándar de vídeo *Microsoft Video for Windows*

3.9 Para saber más

La única y mejor forma de aprender a manejar y configurar un sistema operativo es practicar, practicar y practicar. Es decir, que realmente lo tengas que usar todos los días y trates de resolver los problemas que te van surgiendo, posiblemente con ayuda más experta. Sin embargo, si aún así se necesita algo de bibliografía para extender lo comentado en este capítulo, dos posibles títulos sobre MS-DOS y Unix son los siguientes:

- ✓ O'Day K. Introducción al MS-DOS, Anaya multimedia, Madrid, 1988.
- ✓ Morgan R., MacGilton H., Introducción al UNIX sistema V, MacGraw-Hill, México, 1990.

4. Redes e Internet

4.1 Introducción

En este capítulo se van a presentar los conceptos fundamentales de las redes de ordenadores y de Internet. En primer lugar se describirán las ideas más relevantes dentro de las redes de ordenadores: qué son, relación con otras disciplinas, principales topologías, protocolos, ... En la segunda parte del capítulo se hablará de Internet. Se comenzará con un repaso a la historia de Internet y después se verán aspectos algo más técnicos así como los servicios que ofrece y la gestión de nombres.

4.2 Las redes de ordenadores

4.2.1 Definiciones básicas

Las redes de ordenadores forman parte de un concepto más amplio: las redes de telecomunicaciones. Según el Anexo de la Ley General de Telecomunicaciones (11/98), una red de telecomunicaciones "está formada por los sistemas de transmisión y, cuando proceda, los equipos de conmutación y demás recursos que permitan la transmisión de señales entre puntos de terminación definidos mediante cable, medios ópticos o de otra índole".

Tradicionalmente las redes de telecomunicación, independientemente del criterio que sigamos para clasificarlas, se suelen dividir en *redes de voz* y *redes de datos*. No obstante, esta división cada vez se utiliza menos ya que la tendencia actual es trabajar con sistemas (redes) digitales y en este caso se trata de la misma forma toda la información, ya sea voz, datos, imágenes, ...

Nosotros sólo vamos a ocuparnos de las redes de ordenadores, la definición más sencilla es:

> red de ordenadores: "un sistema de ordenadores autónomos interconectados entre sí y compartiendo recursos"

La *Figura 4-1* presenta un ejemplo sencillo de red.

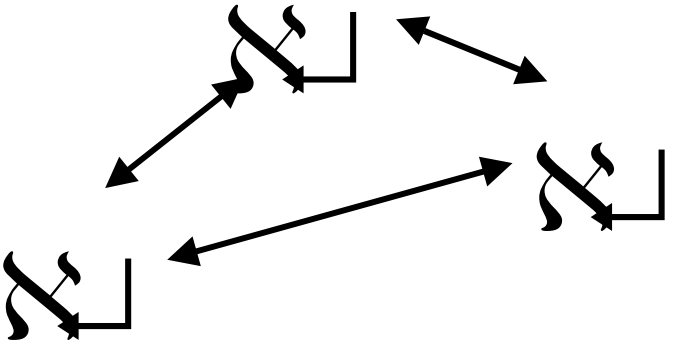


Figura 4-1. Ejemplo sencillo de red

El concepto de ***sistemas distribuido*** se relaciona con el de red de ordenadores, pero son dos cosas diferentes con ciertos puntos de unión. Los sistemas distribuidos son aquellos sistemas en los que las funciones propias de un sistema informático están repartidas entre varios procesadores. En la práctica es difícil separar el concepto de red y el de sistema distribuido.

Las redes de ordenadores se utilizan para transmitir información a través de ellas. Una de las principales aplicaciones de las redes de ordenadores es la transmisión de datos a través de ellas. Podemos definir la transmisión de datos como la parte de la transmisión de información que consiste en el movimiento de información codificada, de un punto a uno o más puntos, mediante señales eléctricas, ópticas, optoelectrónicas o electromagnéticas. Por ejemplo, cuando hacemos un *ftp* para obtener un fichero (por ejemplo de texto), nosotros vemos en la pantalla del ordenador el resultado, pero esa información se ha transmitido en forma de señal eléctrica. Otro ejemplo sería la transmisión de voz por una línea telefónica. En este caso es necesario transformar una señal acústica en una señal eléctrica y codificar el contenido de forma que se transmita correctamente por el medio de transmisión.

Una vez introducido el concepto de red de ordenadores y su relación con otros elementos y disciplinas, podemos pasar a estudiar los diferentes elementos de una red y su clasificación.

4.2.2 Clasificación

Los criterios en base a los que se puede realizar la clasificación de una red son muy diversos. Vamos a considerar la clasificación según la cobertura y según el objetivo empresarial.

Según la cobertura:

- LAN (Local Area Network): redes de área local. Pueden abarcar desde unos pocos metros hasta cubrir un edificio, o como máximo unos edificios cercanos entre sí (por ejemplo, el entorno de un campus universitario).
- WAN (Wide Area Network): redes de área ancha. Es la red de mayor cobertura, llegando a cubrir el área de todo un país, continente o incluso más.
- MAN (Metropolitan Area Network): redes de área metropolitana. Abarcan un área intermedia entre las LAN y las WAN.

Según el objetivo empresarial:

- Públicas. Son accesibles por cualquier usuario siempre que se pague el precio establecido para la conexión. La mayoría son redes WAN.
- Privadas. Las redes privadas habitualmente pertenecen a empresas o instituciones. Sólo son utilizadas por los trabajadores y el objetivo es mejorar la comunicación dentro de la empresa.

4.2.3 Elementos de una red

Una red de ordenadores (Figura 4-2) está formada por dos elementos:

- Host
- Subred de comunicaciones

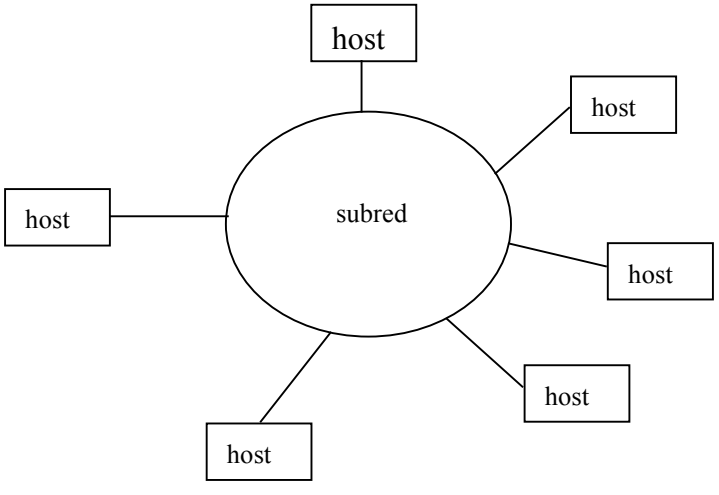


Figura 4-2. Elementos de una red

De forma simplificada podemos definir el ***host*** como el ordenador principal de un organismo o empresa. Debe ejecutar procesos para dar servicio a sus usuarios. También debe ser capaz de comunicarse con otros hosts a través de la subred.

La ***subred de comunicaciones*** es el entramado o estructura al que se conectan los hosts. Debe garantizar la conectividad total, es decir, que dos hosts cualesquiera se puedan comunicar. Está formada por:

- Nodos: elementos que conmutan (pasan) las señales de unas líneas a otras.
- Líneas: medio por el que se transmiten las señales.

La conectividad total no significa que todos los hosts tengan que estar conectados físicamente entre sí, sino que se pueda transmitir información entre cualquier par de ellos. Un ejemplo de conectividad total es la red de carreteras que nos permite viajar entre cualquier par de puntos, pero no hay un caminodedicado para unir cualquier par de ciudades o pueblos.

4.2.4 Topología

Ahora ya sabemos qué es una red de ordenadores; el siguiente paso será conocer cómo podemos realizar las conexiones entre los diferentes elementos: conectar todos los nodos entre sí o usar un cableado principal al que se van añadiendo nuevos elementos a medida que aumenta el número de nodos, ... Las diferentes formas en las que esto se puede realizar es lo que denominamos topología.

La ***topología*** es la disposición física de los distintos elementos que componen una red, con indicación de los medios de enlace utilizados entre los nodos.

Al plantear una red se utilizan dos tipos de líneas:

- Líneas punto a punto (*Figura 4-3*): conecta dos nodos directamente, sin intermediarios.

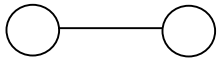


Figura 4-3. Línea punto a punto

- Líneas multipunto (*Figura 4-4*): la línea es un recurso compartido entre varios nodos. Cada vez que un nodo transmite un mensaje, éste se transmite a todos los demás.

La clasificación de las topologías se hace en base a los dos tipos de líneas que acabamos de explicar. En la realidad, existen topologías mixtas.

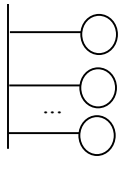


Figura 4-4. Línea multipunto

Topologías punto a punto

Topología en estrella

Hay un nodo especial conectado a todos los demás. Los nodos periféricos sólo están conectados al nodo central. Es una topología adecuada cuando hay mucho tráfico entre el nodo central y los periféricos. Es muy sensible a fallos: si el nodo central cae, toda forma de comunicación se rompe.

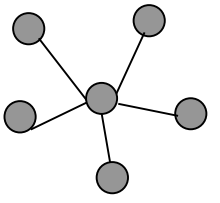


Figura 4-5

Topología en árbol

Se puede considerar como la interconexión jerárquica entre varias estrellas. Se utiliza en redes con un elevado número de usuarios y en redes WAN.

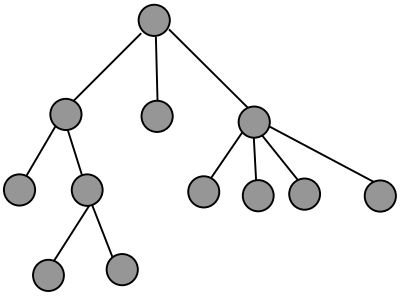


Figura 4-6

Topología en lazo o bucle

Hay *n* nodos y *n* líneas. Cada nodo está conectado a otros dos formando un circuito cerrado. Si hay un fallo en una línea no se pierde la conectividad.

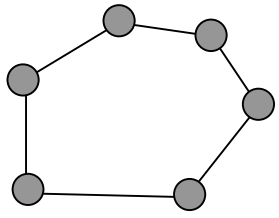


Figura 4-7

Topología completa

Cada nodo está conectado a todos los demás. Se necesita un número elevado de líneas. Se utiliza está topología en redes pequeñas y cuando es importante garantizar la conectividad entre todos los nodos frente a posibles fallos.

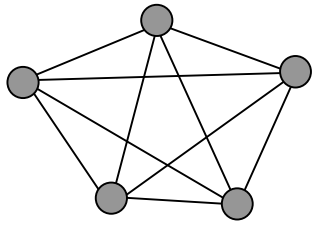


Figura 4-8

Topología irregular

Cualquier conexión que no encaje en la clasificación anterior. Es una topología a medida: a partir de las necesidades de los usuarios se realiza el diseño.

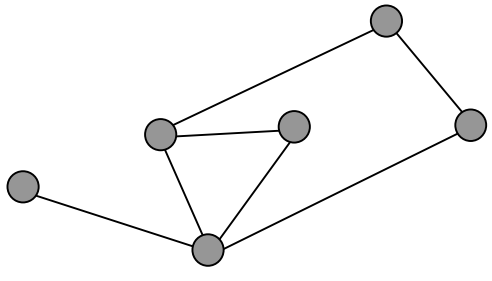


Figura 4-9

Topologías multipunto

Bus

Los elementos de los extremos son los **terminadores**, tienen como función absorber las señales que se transmiten cuando llegan a los extremos del bus

para evitar que se produzcan efectos de reflexión y continuen en la línea de forma indefinida.

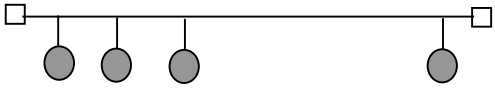


Figura 4-10

Cuando un nodo coloca información (una señal) en el bus, ésta se transmite en ambos sentidos hasta alcanzar los terminadores que la retiran de la línea. De esta forma se garantiza que todos los nodos *han escuchado* la señal. Los nodos tienen capacidad para emitir, recibir y detectar si la línea está ocupada o libre.

Anillo

Es un circuito cerrado. No debe confundirse con un bus al que se unen los extremos. Ahora se debe garantizar que la señal sólo se propague en un sentido.

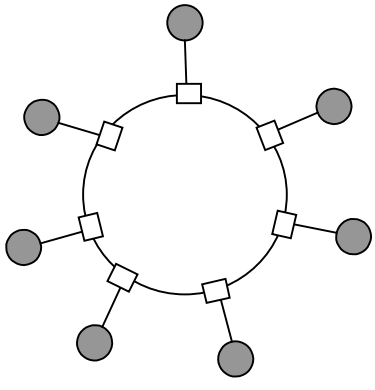


Figura 4-11

Radiodifusión

Es una topología multipunto. El ejemplo típico son las transmisiones de radio. Lo más novedoso con respecto al bus o al anillo es que aquí siempre el medio de transmisión es el aire.

4.2.5 Conmutación

La conmutación es una idea central en la transmisión de información. Siempre que no exista un camino directo entre emisor y receptor será necesario aplicar alguna técnica de conmutación para que la información se transmita correctamente entre emisor (origen de los datos) y receptor (destinatario de la transmisión). Si se utiliza una red con topología multipunto, hay un camino

directo entre emisor y receptor, pero en el caso punto a punto, no siempre existirá este camino y se aplicarán técnicas de conmutación.

Hay tres técnicas de conmutación:

- Circuitos
- Mensajes
- Paquetes

Para explicarlas vamos a hacerlo sobre un ejemplo. Sea una red como la de la Figura 4-12.

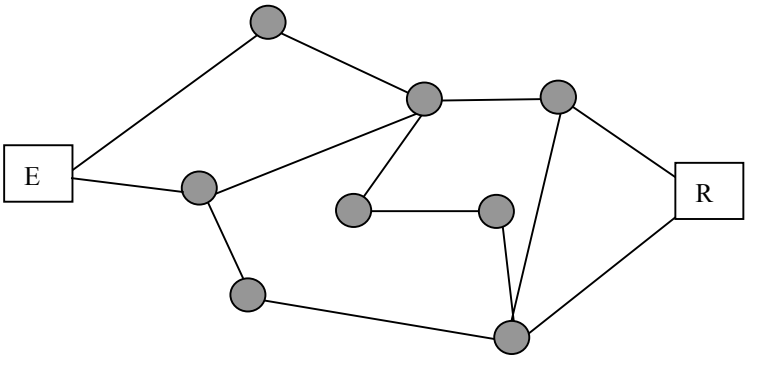


Figura 4-12

Queremos comunicar el emisor (E) con el receptor (R) y no tenemos un camino directo entre ambos.

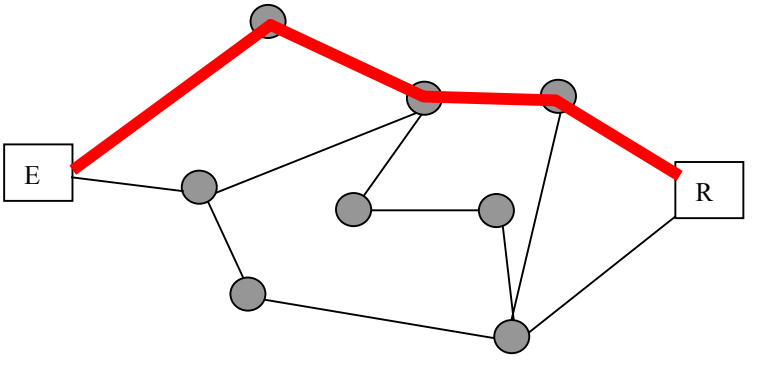


Figura 4-13. Conmutación de circuitos

En la ***conmutación de circuitos*** (Figura 4-13) se establece un circuito entre el emisor y el receptor. Una vez establecido este camino, estará activo mientras dure la comunicación y el resto de la red es como si no existiese. Un ejemplo típico: las comunicaciones de teléfono. Una vez marcado el teléfono del receptor se establece el camino: tiempo en el que no escuchamos nada en la línea telefónica una vez marcado el número. Una vez establecido el camino, se

escuchan los pitidos típicos hasta que el receptor contesta la llamada. Ese camino será "nuestro" mientras dure la comunicación.

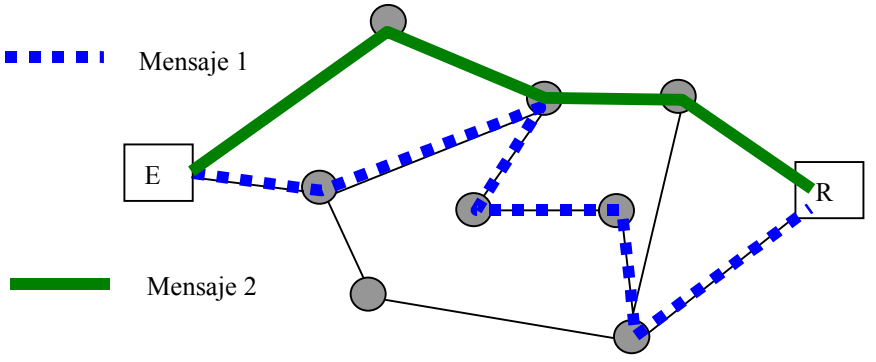


Figura 4-14. Conmutación de mensajes

La técnica de ***conmutación de mensajes*** (Figura 4-14) se desarrolló para las transmisiones entre equipos informáticos. La información se transmite en *mensajes* que pueden tener cualquier tamaño. Unos mensajes serán muy pequeños y otros muy grandes. Por ejemplo, si alguien quiere transmitir un fichero de 2000K entre dos ordenadores, eso sería un mensaje. Si a los pocos minutos quiere transmitir otro de 20K, sería otro mensaje diferente.

Lo interesante de esta técnica es que cada mensaje puede seguir un camino diferente en la red. Es decir, no se establece un camino que está activo durante toda la comunicación, sino que cada mensaje de una misma transmisión sigue un camino y la decisión de ir por un sitio u otro viene determinada por el tráfico que haya en cada nodo (entre otros parámetros).

La ***conmutación de paquetes*** se diferencia de la de mensajes en que los paquetes tienen un tamaño máximo permitido: si lo que se quiere transmitir supera un determinado tamaño, hay que dividirlo en bloques más pequeños, denominados ***paquetes***. En cada paquete es necesario incluir una cabecera donde se indique "el número de paquete", para que el receptor sea capaz de ordenar la información.

4.2.6 Medios de transmisión

Los medios de transmisión son el material del que está formado/fabricado la línea por la que se transmite la señal. Se pueden clasificar en:

- Guiados: hay un soporte físico (por ejemplo, fibra óptica).
- No guiados: el medio de transmisión, habitualmente, es el aire.

Cuando se plantea una red, pueden utilizarse varios medios de transmisión, por ejemplo, una parte del trazado de la red de telefonía fija está realizado con fibra óptica y en otra se usa como medio de transmisión el aire (enlaces por microondas).

Medios de transmisión guiados

Dentro de los medios de transmisión guiados distinguimos:

- Conductor metálico: par trenzado y cable coaxial.
- Conductor óptico: fibra óptica.

El término hilo y cable se usan indistintamente cuando se habla del cableado de las redes, pero tienen unas diferencias que conviene aprender aunque en el lenguaje cotidiano las olvidemos:

- El *hilo* es una hebra de cobre que transporta/transmite la corriente a un determinado voltaje.
- El *cable* es todo el conjunto: el hilo al que se le ha añadido el aislante, los conectores y el soporte para que tenga cuerpo.

El ***par trenzado*** es un tipo de pares de cable, está formado por dos hilos (cada uno de ellos rodeado de aislante) que están de principio a fin trenzados. Aunque la calidad es peor que cuando se usan cables coaxiales o fibra óptica, si las distancias de transmisión son pequeñas los resultados son buenos. El ancho de banda va desde 1 a 20 MHz. Es el medio utilizado para el cableado telefónico dentro de las casas por ser un medio de transmisión barato.

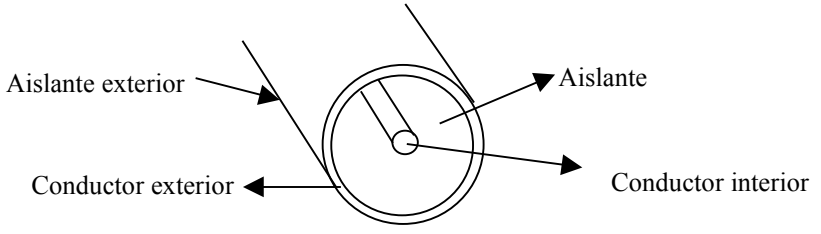


Figura 4-15. Cable coaxial

El ***cable coaxial*** (Figura 4-15) está formado por dos conductores de cobre o de aluminio, uno interior (macizo), separado del segundo conductor por un material aislante. Todo ello está recubierto por un aislante exterior. El ancho de banda es mayor y presenta menos problemas de atenuación[8] que el par trenzado. La señal de televisión se transmite desde la antena a los receptores a través de un coaxial. También se usan en telefonía fija para la unión de centrales y para cableado submarino. Es habitual agrupar varios cables coaxiales para formar cables mayores.

La ***fibra óptica*** es el medio más nuevo para transmitir grandes cantidades de datos. Se basa en principios físicos conocidos desde el siglo XIX, pero como

[8] La **atenuación** es el rozamiento de las señales. A medida que se transmiten las señales van perdiendo potencia hasta que se mezclan con el ruido y no es posible entenderlas. La atenuación depende del medio de transmisión.

no existía la tecnología adecuada no se pudo utilizar hasta mucho después. La fibra óptica está formada por un núcleo de vidrio o plástico y un revestimiento que mantiene la luz en su interior. Entre sus ventajas podemos citar:

- Gran ancho de banda con lo que se logran transmisiones a más de 1 Gbit/seg.
- Muy poca atenuación.
- El material del que se construye es el sílice: hay mucho y barato.
- Es casi inmune a las radiaciones externas, es decir, no presenta problemas de diafonía.

Actualmente el coste de fabricación de la fibra óptica ha disminuido mucho lo que ha llevado a su amplia utilización y a que reemplace a los cables coaxiales en muchas redes.

Medios de transmisión no guiados

En los medios de transmisión no guiados se utiliza el aire para transmitir las señales. Normalmente se transmiten señales vía radio.

En la técnica del ***enlace de microondas*** el emisor y el receptor disponen de sendas antenas parabólicas que deben tener un enlace visual. Este tipo de enlace se utiliza en las comunicaciones telefónicas entre ciudades.

El inconveniente de este tipo de transmisión es que se necesita un enlace visual entre emisor y receptor, lo que sólo puede conseguirse en distancias cortas. Una variante de esta técnica consiste en el uso de ***satélites de comunicaciones***: la transmisión se origina en un solo punto, desde el emisor (en la superficie terrestre) se envía la señal hacia el satélite, que la reenvía a otro punto en la superficie terrestre. Los satélites describen órbitas alrededor de la tierra. Muchos de ellos describen órbitas geoestacionarias (36000 Km sobre el Ecuador).

4.2.7 Arquitectura de redes

Los equipos conectados o que pertenecen a una determinada red son capaces de realizar unas funciones. Al modelo que representa las funciones realizadas por los equipos, lo denominamos ***arquitectura de redes***.

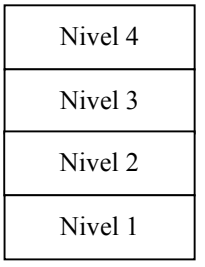


Figura 4-16

Una arquitectura de red queda definida una vez que se definen las funciones a realizar. Pero es necesario agrupar esas funciones siguiendo algún criterio. Normalmente se construyen bloques con las funciones que realizan tareas parecidas. Estos bloques reciben el nombre de ***niveles*** (Figura 4-16). Los niveles se suelen nombrar con números: nivel 1, nivel 2, ... Un número más bajo hace referencia a un nivel cuyas funciones son de más bajo nivel, por ejemplo, el nivel 1 se ocupa de colocar las señales en la línea de transmisión y el nivel 4 puede ocuparse de dividir la información en trozos más pequeños. No todas las arquitecturas de red tienen el mismo número de niveles.

Las ***entidades*** son los elementos que realizan las funciones de cada nivel. Cuando se define una arquitectura lo que se debe especificar claramente son las funciones, las entidades son dependientes del fabricante, es decir, un fabricante puede utilizar tres entidades para realizar las funciones de un nivel y otro puede usar cuatro, uno puede hacerlo vía software y otro vía hardware.

El ***protocolo*** (Figura 4-17) tiene como objetivo que dos niveles equivalentes de usuarios diferentes puedan entenderse. El protocolo es el conjunto de normas que regulan la comunicación entre niveles equivalentes en usuarios diferentes.

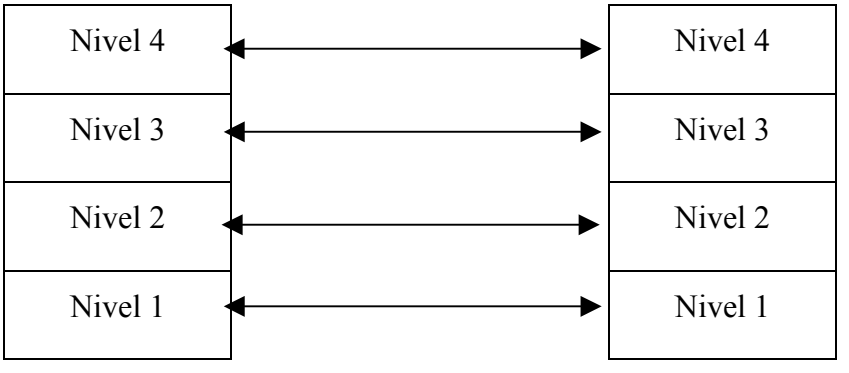


Figura 4-17. Protocolos entre niveles

La definición de protocolo no es mas que una extensión al caso de las redes de ordenadores del concepto que manejamos en el lenguaje coloquial.

La función de cada capa (nivel) es proporcionar ***servicios*** a la capa que está por encima de ella. El concepto de servicio y el de protocolo se confunden con mucha frecuencia.

> servicio: "un conjunto de operaciones que ofrece una capa a la que está por encima de ella"

> protocolo: "un conjunto de reglas que definen el intercambio de información entre niveles pares de usuarios diferentes"

Si pensamos en ejemplos de la vida real, el protocolo serían las reglas que permiten que dos personas que pertenecen a la misma profesión pero en países diferentes, se puedan comunicar.

Los ***servicios*** pueden ser ***orientados a conexión*** o ***sin conexión***. Los explicaremos mediante un ejemplo. En los servicios orientados a conexión primero se establece la conexión entre emisor y receptor, después se transmite toda la información y por último se liberan los recursos utilizados. Esto es lo que sucede cuando hacemos llamadas telefónicas: primero marcamos el número (se establece el camino), después hablamos y por último colgamos (se libera la línea: los recursos). En un servicio sin conexión cada mensaje (trozo de información a transmitir) debe contener la dirección del destinatario. Esto es lo mismo que sucede cuando enviamos cartas: aunque enviemos 2 cartas a la misma persona es necesario que escribamos la dirección del remitente en ambos sobres o de lo contrario no alcanzarían su destino.

Modelo de referencia OSI

El modelo de referencia OSI (Open Systems Interconnection) fue desarrollado en la década de los 80 en la ISO (International Standarization Organization). La traducción de OSI al español es Interconexión de Sistemas Abiertos, es decir, se trata de un modelo para la conexión de equipos que están *abiertos* a la comunicación con otros sistemas.

Desde su aparición, este modelo ha sido explicado de forma exhaustiva en numerosos libros de redes de ordenadores y sistemas de comunicación ([TAN], [STA], [HUI], ...). De todos ellos, quizás el más popular sea el Tanenbaum y es el que tomaremos como referencia para la explicación.

El modelo OSI tiene siete niveles, aunque no todos ellos son necesarios para que se realice la comunicación con éxito. Sólo el nivel 1, 2 y 7 son imprescindibles, y si se quieren realizar determinadas operaciones sería recomendable una versión reducida del nivel 4.

Para referirnos a los niveles podemos usar su nombre o un número, así el nivel 7 es el nivel de aplicación, el 6 el de presentación, el 5 el de sesión, ...

Los tres primeros niveles (físico, enlace y red) incluyen todas las funciones para que sea posible la comunicación dentro de la subred de comunicaciones.

El modelo OSI no es una arquitectura de red porque sólo dice lo que se debe hacer en cada nivel, pero no cómo hacerlo. Es decir, no especifica los servicios y protocolos que se deben utilizar en cada nivel. A continuación veremos brevemente cada uno de los niveles.

El ***nivel físico*** (1) se ocupa de la transmisión de bits. A este nivel le llega la información transformada por los niveles superiores, aquí ya no se transforma más. Se decide los pulsos de reloj necesarios para transmitir un bit, el nivel de tensión para representar al bit 0 y al 1. Es decir, este nivel se ocupa de lo que se denomina sincronismo de bit.

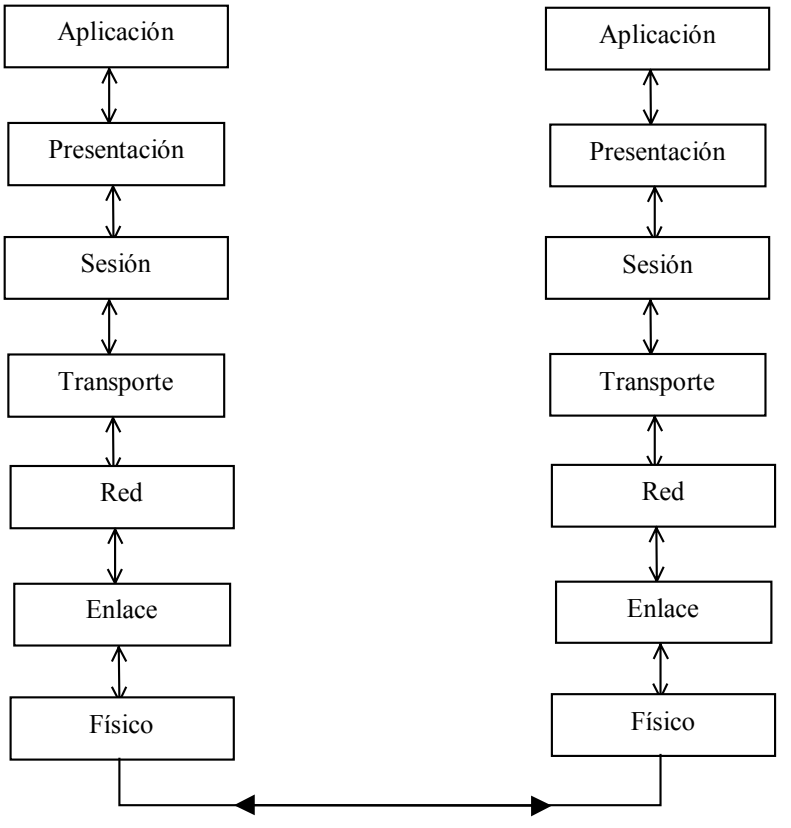


Figura 4-18. Niveles OSI

El ***nivel de enlace de datos*** (2) añade una cabecera y una cola a los bloques de datos que le pasa el nivel de red, el resultado son las tramas. El nivel físico se limita a transmitir los datos como una ristra de bits, sin tener en cuenta ningún mecanismo de control de errores. El nivel 2 va a encargarse de introducir mecanismos de control de errores que aseguren que la información se recibe

correctamente en el receptor y cuando haya problemas, se retransmitan las tramas necesarias.

El ***nivel de red*** (3) se ocupa de controlar el funcionamiento de la subred. A los datos que se reciben del nivel 4 se les añade una cabecera, el resultado es el *paquete*. Este nivel es el encargado de decidir el camino que seguirán los paquetes por la subred, es decir, el encaminamiento. Además debe ser capaz de dar soluciones cuando un paquete pasa de una red a otra y en la nueva red no se acepta el paquete por ser demasiado grande o porque la dirección de la cabecera no está en un formato que entienda.

El ***nivel de transporte*** (4) acepta los datos del nivel de sesión, dividiéndolos en unidades más pequeñas, si es necesario, dando como resultado los *mensajes* que son pasados a la capa de red. Además se asegura que los mensajes lleguen correctamente al destinatario. Se realiza un control de errores, pero de forma diferente al nivel 2.

El ***nivel de sesión*** (5) permite a dos usuarios de máquinas diferentes establecer sesiones entre ellos. Una sesión puede consistir en la transferencia de un fichero. Este nivel se encarga de la sincronización: si estamos realizando una transferencia de ficheros que dura mucho tiempo, puede romperse la comunicación. Una vez restablecida, para que no sea necesario volver a transmitir todo el fichero, lo que hace el nivel de sesión es insertar unos puntos de verificación que permiten que sólo se realice la retransmisión de los datos a partir del último punto de verificación.

El ***nivel de presentación*** (6) se ocupa de la sintaxis y de la semántica de la información que se transmite, no sólo de mover bits de forma fiable. Se va a ocupar de la codificación de los datos según lo acordado previamente. Este nivel actúa como traductor.

El ***nivel de aplicación*** (7) actúa como interface entre el usuario y el resto del modelo. El usuario *no sabe* lo que sucede por debajo del nivel 7.

Transmisión de datos en el modelo OSI

Veamos mediante un ejemplo (Figura 4-19) cómo se pueden transmitir datos utilizando el modelo OSI ([TAN]).

Sean dos usuarios A y B. El usuario A tiene cierta información que desea transmitir a B, así que entrega los datos al nivel de aplicación, que le añade la cabecera[9] propia de esta capa y que vamos a llamar *AH*. Esta cabecera puede ser nula. El elemento resultante se entrega al nivel de presentación. Este nivel puede realizar diferentes transformaciones (por ejemplo, puede comprimir los

[9] Cabecera en inglés es *header*. La *h* se suele usar como inicial para referirse a la cabecera. Cola en inglés es *tail* y la *t* se usa como inicial para referirse a cola.

datos) a los datos recibidos, e incluso añadir una cabecera (*PH*), entregando el resultado al nivel de sesión. Es importante remarcar que el nivel de presentación no sabe qué porción de los datos que recibe de la capa de aplicación corresponden a la cabecera (¡si es que existe!) y cuál a los datos.

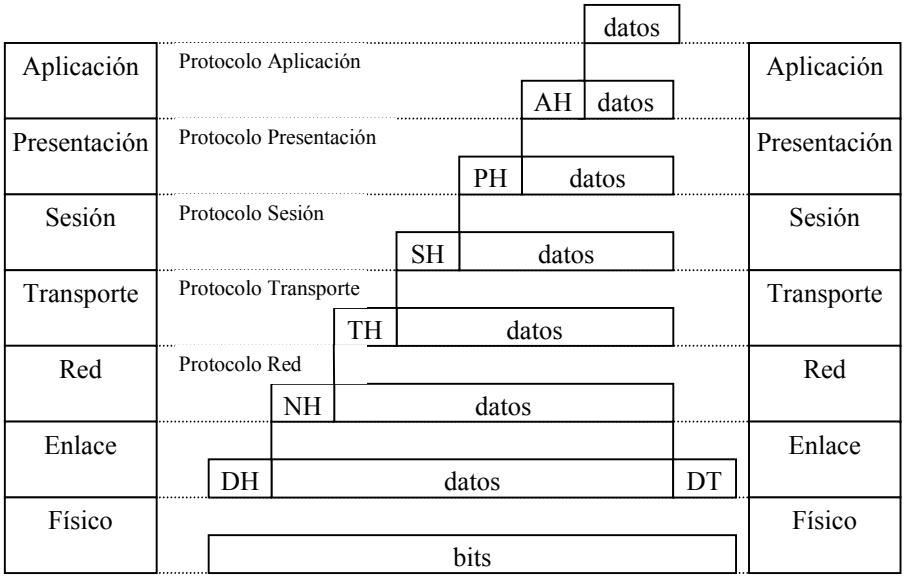


Figura 4-19. Ejemplo de transmisión de datos

El nivel de sesión puede añadir una cabecera (*SH*) y pasa los datos al nivel de transporte. Esta capa divide el mensaje en unidades más pequeñas, si es necesario, y les coloca una cabecera (*TH*). Esta división suele hacerse porque el nivel 3 suele tener limitado el tamaño de los elementos.

La capa de red añade sus propias cabeceras (*NH*[10]) después de decidir qué líneas van a utilizarse para la transmisión. La capa de enlace de datos (nivel 2) añade una cabecera (*DH*) y una cola (*DT*) y entrega el resultado al nivel físico que se encarga de la transmisión de los datos al receptor B. En el receptor se retiran los encabezados uno a uno a medida que los datos se propagan hacia arriba por los diferentes niveles hasta llegar al proceso receptor.

La idea clave en todo este proceso es que aunque la transmisión real de los datos es vertical, desde el punto de vista conceptual los procesos pares (procesos que pertenecen al mismo nivel en máquinas diferentes) conciben su comunicación como si fuese horizontal: de protocolo *n* a protocolo *n*.

[10] NH: Net Header: cabecera de red.

4.2.8 Ejemplos de redes y arquitecturas

En los últimos años, las redes de ordenadores han crecido enormemente. Cada vez hay más usuarios y estos demandan aplicaciones más complejas. El avance de los ordenadores ha permitido que sean posibles aplicaciones multimedia, lo que hasta hace poco era impensable porque, entre otras cosas, se necesita mucho ancho de banda.

En este apartado vamos a comentar brevemente algunas de las redes y arquitecturas más utilizadas, desde las más antiguas hasta las que se están comenzando a implantar en la actualidad. Veremos los conceptos más básicos de X.25, Ethernet, ATM, ...

X.25

X.25 es un estándar para el acceso a redes públicas de conmutación de paquetes desarrollado durante la década de los 70. Conceptualmente se compone de tres niveles. El nivel 3 permite al usuario establecer circuitos virtuales[11] y después enviar paquetes de hasta 128 bytes por ellos.

Telefónica oferta desde hace muchos años un servicio de transmisión de datos, nacional e internacional, basado en el protocolo X.25. Este servicio es IBERPAC.

Ethernet

La IEEE 802.3 se conoce popularmente como Ethernet, es una red de transmisión basada en un bus con control de operación descentralizado a 10 ó 100 Mbps. Los equipos conectados a una red Ethernet pueden transmitir cuando quieran. Si dos o más paquetes chocan, cada equipo espera un tiempo aleatorio y vuelve a intentar la transmisión.

Frame Relay

Frame relay y ATM surgen como la respuesta de los fabricantes a la demanda de los usuarios: multimedia, comunicaciones más rápidas,...

Frame relay es una técnica de conmutación de paquetes simplificada. Utiliza medios digitales de alta velocidad y con una tasa de error muy baja. De esta forma logra transmitir un volumen elevado de datos a velocidades de transmisión muy altas. Podemos ver frame relay como una línea virtual alquilada. Un usuario puede alquilar una línea para su uso exclusivo. Las transmisiones con líneas dedicadas son muy rápidas y caras. Así que una solución es alquilar una línea virtual, el comportamiento será muy parecido al

[11] Un circuito virtual se basa en ideas parecidas a las de la conmutación de paquetes. Se establece una conexión lógica entre los equipos de terminación de datos identificada por el VCI (Virtual Connection Identifier).

de la línea dedicada real siempre que no se exceda el uso promedio de un nivel predeterminado.

RDSI

La Red Digital de Servicios Integrados (en inglés, ISDN: Integrated Services Digital Network) es la evolución de la red de telefonía convencional. Todo el servicio está digitalizado: las líneas, las centralitas,... y se integran los servicios de voz y datos en el mismo medio.

La RDSI proporciona una serie de servicios integrados como transmisión de voz, transmisión de datos, fax , correo electrónico, internet, televisión, ...

ATM

En ATM (Asynchronous Transfer Mode): Modo de Transferencia Asíncrono, para la transmisión se usan paquetes de tamaño fijo. Estos paquetes se llaman celdas y en ocasiones en vez de hablar de ATM se habla de *cell relay* para hacer una analogía con *frame relay*. Algunas de las ideas son parecidas a las utilizadas en conmutación de circuitos.

Cuando hablamos de la RDSI podemos referirnos a la de banda estrecha o a la de banda ancha. ATM es la tecnología por debajo de la RDSI de banda ancha (RDSI-BA). La RDSI-BA con ATM tiene su propio modelo de referencia que es diferente al OSI.

4.2.9 Interconexión de redes

La interconexión de redes (Figura 4-20) se hace necesaria desde el momento que tenemos a dos usuarios que desean conectarse y que no utilizan la misma red, lo que supone que van a usar protocolos y tecnologías diferentes.

Por ejemplo, queremos transmitir información entre dos usuarios, uno de los cuales usa X.25 y el otro Ethernet, ¿qué hacer?.Los problemas que nos podemos encontrar son muy variados, desde servicios diferentes, velocidades diferentes, medio físico distinto, ...

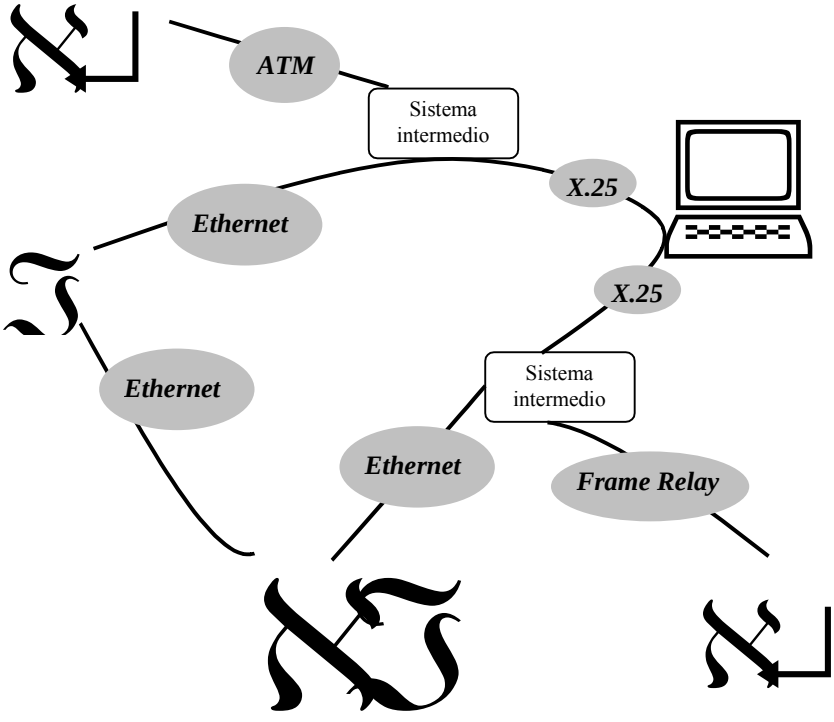


Figura 4-20. Interconexión de redes

La forma de resolver el problema es utilizar unos ***dispositivos intermedios*** que nos permitan conectar las subredes diferentes. Estos nodos intermedios no van a implementar todos los niveles OSI. Habitualmente el nivel más alto que incluyen es el de red, aunque también pueden incorporar el nivel 4 ó el 7.

Dos niveles pares (niveles equivalentes en máquinas diferentes) sólo pueden comunicarse cuando son iguales, si no lo son se hace necesario introducir un dispositivo intermedio. Por ejemplo, supongamos dos subredes, tal que los dos primeros niveles son diferentes y el resto son iguales. De forma gráfica se muestra el resultado en la Figura 4-21.

El nombre que se usa para el dispositivo intermedio depende de la capa que se encargue del trabajo.

Como dispositivo intermedio en el ***nivel 1*** se usan los ***repetidores*** que copian bits entre segmentos de cable. Además de realizar una conversión amplifican la señal. Son útiles cuando tenemos que unir dos redes que, por ejemplo, sólo se diferencian en el soporte físico (una puede usar cable coaxial y la otra fibra óptica) o cuando la distancia es grande.

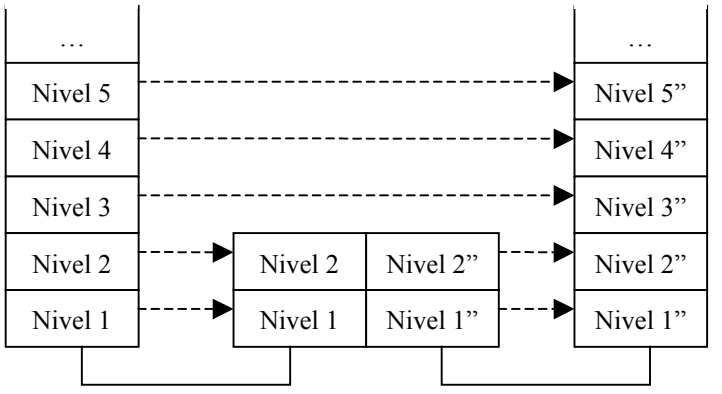


Figura 4-21. Subredes con nivel 1 y 2 diferentes

En el ***nivel 2***, se usan los ***puentes*** (***bridges***) que almacenan y reenvían las tramas con los cambios necesarios en la cabecera. Estos elementos son más complejos que los repetidores. Se componen de una CPU y memoria.

Los ***encaminadores*** (***routers***) son los dispositivos del ***nivel 3***, su tarea es equivalente a la de los puentes, pero ahora se trabaja con paquetes.

Cuando las diferencias entre las redes se producen en las capas por encima de la 3, se usan las ***pasarelas*** o ***gateways***.

4.3 Internet

Cuando hoy hablamos de Internet a lo que nos estamos refiriendo es a un conglomerado de ordenadores de diferente tipo, marca y sistema operativo, distribuidos por todo el mundo y unidos a través de enlaces de comunicaciones muy diversos. Pero, ¿cómo surgió esta red de redes?.

4.3.1 Un poco de historia

A finales de los años sesenta, una de las preocupaciones de las Fuerzas Armadas de los Estados Unidos era conseguir que las comunicaciones estuvieran descentralizadas, es decir, evitar un centro neurálgico de comunicaciones que pudiera ser destruido en un eventual ataque militar con armas nucleares y que así, aún sufriendo el ataque, las comunicaciones no se bloquearan, sino que solamente se perdiera un nodo. De esta forma surgió en 1969 DARPANET. El número de ordenadores que formaban la red fue aumentando poco a poco y en 1972 cuando ya había 41 nodos se cambió el nombre de la red a ARPANET. Justo un año antes, en 1971, se desarrolló el primer programa para enviar correos electrónicos.

Fue en este momento cuando las instituciones académicas se interesaron por estas posibilidades de conexión. La NSF (National Science Foundation) dio acceso a sus seis centros de supercomputación a otras universidades a través

de ARPANET. Se pensó que la red debía ser lo más sencilla posible de forma que los cambios de tecnología afectasen sólo a los ordenadores de los extremos de la red y no al cableado que los unía. La red se encargaría de entregar bien los paquetes que se enviasen, y las tareas más complejas se realizarían en los extremos. Durante los setenta se desarrollaron varios estándares y protocolos (como telnet). Además tanto redes diferentes de ARPANET como organizaciones se conectan a ésta.

En 1983 ARPANET se separa de la red militar que la originó, y se conecta con Csnet y MILnet, dos redes independientes. Esta es la fecha que se suele tomar como el nacimiento de Internet. Como anécdota, también en este año aparece la primera versión de Windows (Microsoft).

Los 80 marcaron el comienzo de la explosión de Internet, pasándose de unos pocos de cientos de ordenadores conectados a más de sesenta mil al final de la década. Los servicios se fueron mejorando y así en 1985, se termina el desarrollo del protocolo para la transmisión de ficheros en Internet (FTP), basado en la filosofía de cliente-servidor. También durante esta década surgen los hackers.

En los noventa, el aumento de usuarios provocó las primeras saturaciones de Internet y para evitar el colapso total fue necesario restringir el acceso. El World Wide Web (telaraña global) desarrollado en el CERN por Tim Berners-Lee sentó las bases del protocolo de transmisión HTTP, el lenguaje de documentos HTML y el concepto de los URL. La WWW fue uno de los métodos para mejorar el funcionamiento.

Todo esto pasaba en el mundo, pero ¿qué ocurría en España?. Hubo que esperar hasta 1991, para que la red IRIS se conectase a Internet y así dar servicio a las universidades.

En 1993 apareció Mosaic, el primer navegador, y la World Wide Web (WWW) comenzó a despuntar. En septiembre de 1993 se inició el primer servidor Web en español. En estos momentos se aumenta la potencia de las redes troncales de EE.UU., y en 1994 se eliminan las restricciones de uso comercial de la red y el gobierno de EE.UU. deja de controlar la información de Internet. Entonces nace la empresa Netscape, y con ella un nuevo navegador: Navigator.

Se suele elegir 1995 como el año del nacimiento de la Internet comercial. Desde ese momento el crecimiento de la red ha superado todas las expectativas. A partir de aquí la escalada de tecnología es impresionante. Se desarrollan los motores de búsqueda que rápidamente añaden búsquedas inteligentes en varios idiomas. El uso del lenguaje Java se extiende. Se desarrolla de una manera definitiva el comercio electrónico, para comprar productos y servicios a través de Internet. Se pueden ver cientos de televisiones y escuchar radios de todo el mundo en tiempo real. Los bancos se

asientan en la Red y la gente empieza a ceder en su miedo inicial, confiando en la seguridad que ofrecen los servidores seguros.

En la actualidad se estima que hay más de 200 millones de usuarios de Internet repartidos en más de 100 países.

Hoy en día Internet no es una simple red, sino miles de redes (unas 100.000) que trabajan en conjunto, empleando un juego de protocolos y herramientas comunes.

¿Qué nos depara en el futuro Internet?. El 15 de Abril de 1998 el vicepresidente de los EE.UU. se presentó ante la prensa para anunciar una revolución "más importante que la invención de la imprenta". Se presentaba Internet2: una red de alta velocidad entre 100 y 1.000 veces más rápida que la actual. Y donde todos los aspectos que en Internet no se han podido desarrollar con éxito vean la luz. Ya hay más de 160 universidades involucradas en el proyecto, así como el gobierno de los EE.UU. y diversas empresas que han aportado mucho dinero para el proyecto.

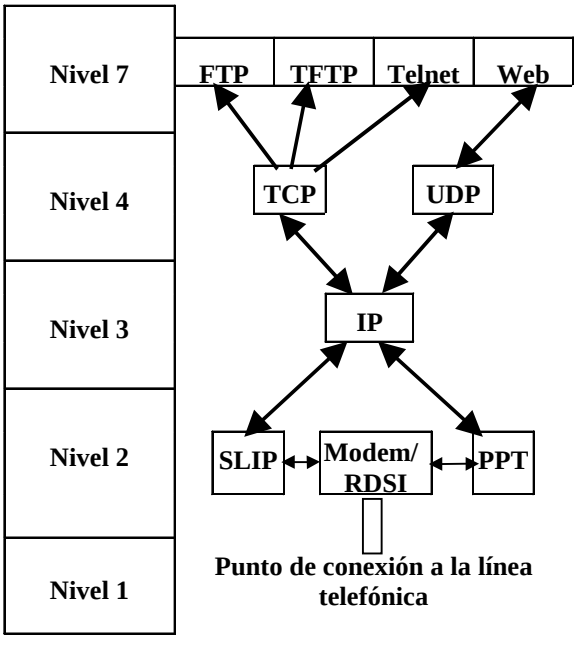


Figura 4-22. Protocolos para el acceso a Internet

4.3.2 Protocolos para acceso a Internet

Un protocolo es un lenguaje de reglas y signos que rigen el intercambio de información entre ordenadores.

TCP/IP son los protocolos utilizados para la transmisión de datos en Internet. Es una familia de protocolos que son la base de la Red. Forman parte de ella

los dos que le dan nombre (IP y TCP) y otros que se describirán brevemente a continuación. En la Figura 4-22 se relacionan los niveles OSI con los diferentes protocolos utilizados para el acceso a Internet.

IP (Internet Protocol)

El protocolo IP define la base de todas las comunicaciones en Internet. Independientemente del tipo de aplicación que se ejecute a través de Internet, éstas se comunican entre sí descendiendo a un nivel inferior a través del protocolo IP, que define el intercambio de datagramas entre el ordenador y los routers. Todo lo que debe ir desde un origen a un destino se fracciona en porciones, más o menos pequeñas, se empaqueta en paquetes IP y, finalmente se transmite. En el nivel IP se crean las bases para que los diferentes tipos de arquitecturas y de hardware que forman la red puedan entenderse.

La comunicación vía IP entre aplicaciones se realiza de un modo totalmente transparente para el usuario, es decir, el usuario no puede ver las gestiones a nivel IP que desarrollan las aplicaciones. Únicamente unos pocos elementos son accesibles al usuario tal y como se han concebido a nivel IP, como por ejemplo la dirección que identifica a cada host formada por cuatro bytes (157.88.32.221).

En el nivel IP se definen los siguientes aspectos de intercambio de información

- Un mecanismo de direcciones que permite identificar al emisor y al receptor.
- Se define un formato para los paquetes. Así los paquetes, en su cabecera, llevan información importante: dirección del emisor, receptor, número de paquete, etc.

UDP (User Datagram Protocol)

Con el protocolo UDP se da más importancia a los datos que se transmiten y a las aplicaciones de Internet que al hardware. Desde este punto de vista los protocolos más importantes son UDP y TCP. Estamos en el nivel de transporte.

UDP entrega datagramas[12] de la misma manera que IP. En ocasiones, los paquetes pueden llegar más rápido de lo que el receptor los puede procesar. Es un tipo de IP pero a nivel de aplicaciones. TCP y UDP trabajan en el mismo nivel, y TCP asegura el orden de entrega de los paquetes, por eso muchas aplicaciones usan TCP y no UDP.

[12] Un datagrama es una unidad de datos que se encamina a través de una red de paquetes mediantes decisiones que se toman en los nodos, sin establecer previamente un camino o circuito.

Sin embargo, el protocolo UDP tiene como ventaja que separa los diferentes canales de comunicación dentro de los host: concepto de puerto.

Con IP y con UDP los paquetes y los datagramas no tienen asegurada la llegada al receptor y en caso de que lleguen pueden hacerlo de forma desordenada o duplicados. Si algún paquete tiene problemas y no llega hasta su destino, el emisor no lo sabrá. Tampoco sabrá si llegan en distinto orden o si llegan corruptos. Ahí es donde entra TCP. Internet necesita un canal de comunicación seguro desde el emisor hasta el receptor y TCP se lo da. Prácticamente todos los servicios de Internet (FTP, HTTP, E-mail...) están basados en TCP.

TCP (Transfer Control Protocol)

Como sucedía con UDP, TCP está pensado para la comunicación entre aplicaciones; por eso el punto final de una conexión TCP viene representado por una dirección IP y un número de puerto. La dirección IP se encarga de hacer llegar los datos al host correcto, y el número de puerto hace que se envíen a la aplicación correcta dentro del host. En un host se pueden establecer varias conexiones TCP simultáneas entre diferentes aplicaciones de Internet y otras aplicaciones del host.

TCP proporciona un canal seguro, por ello el emisor debe tener la posibilidad de saber si el receptor ha recibido correctamente la información. Los paquetes en TCP se componen de una cabecera de 20 bytes y a continuación los datos. Este campo puede tener un tamaño muy pequeño o muy grande.

En resumen, TCP se caracteriza por las siguientes funciones:

- **Servicio Orientado a Conexión:** el receptor recibe exactamente la misma secuencia de bytes que envía la máquina origen.
- **Conexión de Circuito Virtual :** durante la transmisión, los dos extremos se comunican para verificar que los datos se reciben correctamente. Cuando se detecta un fallo se ponen en marcha los mecanismos adecuados para solucionarlo.
- **Transferencia con Memoria Intermedia:** para transmitir la información se pueden usar bloques de datos de diferentes tamaños, se elige el que sea mejor para la transmisión. Esos datos que se desean transmitir se dividen o agrupan para formar los paquetes que serán transmitidos. El protocolo utilizado es el que decide el tamaño de lo que se transmite. En el receptor se eliminan las cabeceras y la información redundante y se entrega la información en el mismo orden en que ha sido enviada.
- **Flujo no estructurado :** Posibilidad de enviar información de control junto a datos.

- **Conexión Full Duplex :** Se permite la transmisión simultánea en ambas direcciones.

HTTP (Hyper Text Transfer Protocol)

HTTP es el *lenguaje* o *idioma* que utilizan tanto los servidores como los clientes Web para comunicarse entre sí. Este protocolo es la base de toda comunicación desarrollada en la Web. Se utiliza desde principios de los 90. HTTP es un protocolo ASCII que se ocupa de establecer una comunicación TCP segura entre el cliente y el servidor siempre que se tenga que producir un intercambio de datos.

Al igual que otros muchos protocolos, los mensajes que se transmiten usando HTTP constan de dos partes: la cabecera y el cuerpo (datos) separados entre sí por una línea en blanco.

Cuando un cliente necesita una información, por ejemplo un documento, envía un mensaje HTTP a un ordenador con servidor Web. Este responde con un documento HTML.

SLIP (Serial Line Internet Protocol)

Para conectarse a Internet, existen dos posibilidades: pertenecer a una red local, en la que los ordenadores están físicamente conectados entre sí, que tiene conexión directa a Internet, o bien conectarse vía módem con una red local con acceso a Internet. Este es el caso de los clientes de los proveedores de acceso a Internet. Pues bien, en este caso en que la conexión se produce vía módem se utiliza el protocolo SLIP o el PPP (su uso está más extendido).

PPP (Point to Point Protocol)

PPP es un protocolo mucho más completo que SLIP. Además de potente, es adaptable y se puede utilizar siempre que se necesiten conexiones punto a punto, ya sea para conectar dos redes locales, el acceso remoto a otro ordenador o acceder a Internet a través de un servidor. Tampoco importa el tipo de conexión ni el hardware utilizado.

4.3.3 Estructura

En este apartado vamos a estudiar cómo se realiza la conexión de un ordenador a Internet. Por una parte está la interconexión física de los equipos y por otra los protocolos de comunicación que permiten que ordenadores muy diferentes puedan entenderse entre sí.

- Mediante una red de área local, por lo general basadas en Ethernet.
- Enlaces nacionales, con líneas de uso exclusivo o compartidas (de una compañía telefónica).

- Enlaces internacionales, proporcionados por compañías de comunicaciones con implantación internacional. Pueden utilizar cableado convencional, fibra óptica, satélites, enlaces por microondas, …
- Mediante módems. Posiblemente sea la opción más empleada. Sobre todo para usuarios particulares. Se conectan a través de una llamada telefónica a un proveedor de comunicaciones que da acceso a Internet. El uso de líneas RDSI (Red Digital de Servicios Integrados) es cada vez más frecuente, como solución de futuro para conectar a usuarios particulares a las redes de información de alta velocidad.
- ADSL (Asymmetric Digital Subscriber Line). Es una línea digital de abonado asimétrica. Esta tecnología permite la transmisión de señales tanto analógicas como digitales utilizando par de cobre trenzado. Cuando la transmisión se realiza hacia el usuario la velocidad de transmisión es mucho mayor que en sentido contrario (es el usuario el que envía datos o peticiones hacia el exterior).
- El cable permite al usuario disponer a través de la misma línea de Internet, telefonía fija y televisión. Se pueden contratar los tres servicios o sólo los que interesen. Para poder acceder a Internet mediante el cable es necesario disponer de un módem (diferente al módem que permite acceder a la red de telefonía).

Independientemente del método que se use para acceder a Internet, se debe garantizar el correcto funcionamiento. Por eso, por encima de todos los usuarios se ejecuta un programa de gestión de comunicaciones que permite entenderse con las demás máquinas mediante el protocolo adecuado.

Como ejemplo veamos qué es lo que sucede cuando intentamos visualizar una página Web. El primer paso consiste en llamar al proveedor de Internet. En esta etapa se produce la conexión de nuestro ordenador con el del proveedor. El protocolo TCP/IP empaqueta los datos y se empiezan a transmitir por el puerto serie. En el receptor, los datos se procesan para poder ser entregados con éxito. Esto es posible porque nuestro proveedor nos asigna una dirección IP entre las muchas que tiene. Cuando ya estamos conectados con el proveedor y le hemos mandado el mensaje, el proveedor intentará saber en qué lugar del mundo se encuentra el ordenador que almacena la información que nosotros buscamos, es decir, cuál es su dirección IP. Para ello en el ordenador del proveedor está en marcha el servidor DNS (*Domain Name Server*). Este programa averigua la dirección IP y se lo comunica al navegador. Entonces aparece en nuestro navegador: "Buscando sitio..." en la barra de estado. Cuando el DNS no puede encontrar por sí mismo la dirección IP, recurre a otro servidor DNS.

Después de comunicarle la información al navegador, se cambia al protocolo HTTP (*Hyper Text Transfer Protocol*) y se busca un servidor HTTP activo,

que corresponda a la dirección transmitida: "Conectando con el sitio...". Para llegar hasta el servidor HTTP, los routers van enviando los paquetes de datos de una red a otra hasta alcanzar el ordenador destino. Como Internet es una red descentralizada, no podemos conectarnos con el ordenador principal y que este nos lleve al que buscamos, sino que debemos ir conectándonos progresivamente con varios ordenadores, uno tras otro, para poder alcanzar la red en la que se encuentra nuestro objetivo, y dentro de la red el ordenador que buscamos. Todo esto se realiza de manera transparente al usuario.

4.3.4 Servicios en Internet

Una de las razones que han propiciado el rápido crecimiento de Internet es la variedad de servicios disponibles y la facilidad de acceso a los mismos. Se distinguen dos tipos de servicios o aplicaciones:

- Servicios básicos, son aquellos sobre los que se apoyan generalmente el resto de aplicaciones o se utilizan para actividades de administración y control de la red.
- Aplicaciones de usuario final: programas concebidos para ser utilizados por el usuario final como el ftp, correo electrónico, news, ...

A continuación se describen los principales servicios.

World Wide Web

La World Wide Web (WWW) o teleraña es el servicio estrella de todos a los que se puede acceder, además de ser el que experimenta un crecimiento mayor. Fue desarrollado en el CERN (Centro Europeo de Estudios Nucleares, Suiza) por Tim Barnes-Lee en 1992.

Mediante este servicio, el usuario dispone de un fácil acceso a la información ofrecida por multitud de servidores, repartidos por todo el mundo. Esta información se puede presentar tanto como texto, gráficos, sonidos, animaciones o vídeo.

La Web se basa en:

- HTTP como mecanismo de entendimiento entre el programa cliente y el programa servidor. Se usa desde 1990.
- HTML (Hypertext markup Language) como lenguaje de descripción y formato de las páginas del Web. Sigue un modelo de desarrollo abierto y está en constante evolución.
- URL (Uniform Resource Locator) como mecanismo estandarizado para dar nombre a las páginas y elementos del Web, asignándoles un título y una ruta de acceso unívocos. Se compone de tres partes: método de acceso, nombre del host y ruta de acceso

Correo electrónico

El correo electrónico o e-mail (electronic mail) permite la comunicación personal entre todos los usuarios de la red. Este servicio es uno de los más utilizados por:

- Rapidez: es mucho más rápido que el correo normal
- Facilidad de uso
- Está disponible en cualquier momento y
- Es barato

Cada usuario está identificado con su dirección de correo:

nombre@nombre_dominio

por ejemplo: prueba@autom.uva.es representa al usuario prueba que pertenece al dominio del departamento de Ingeniería de Sistemas y Automática de la Universidad de Valladolid (España).

Uno de los programas de correo más conocidos para entornos Windows es Eudora. Trabaja con el protocolo POP (Post Office Protocol) para traer los correos y con el SMTP (Simple Mail Transfer Protocol) para enviarlos.

Hay empresas que ofrecen cuentas de correo a través de sus páginas web donde se pueden encontrar interfaces muy sencillos de manejar.

Es importante destacar que cuando se envía un correo electrónico no es necesario que la máquina del destinatario esté conectada en ese momento. Además de texto se pueden transmitir imágenes y sonidos. Esta última característica se logra con MIME (Multipurpose Internet Mail Exchange).

Transferencia de ficheros

El envío de ficheros o FTP (File Transfer Protocol) permite transmitir ficheros de cualquier tipo (texto, audio, imagen, ...) entre todo tipo de ordenadores conectados a través de Internet. La primera versión de FTP es de 1971, pero no se puso realmente en funcionamiento hasta 1985 y hoy en día sigue siendo ampliamente utilizado.

Acceso remoto

El acceso a este servicio conocido normalmente como TELNET (Telecommunicating Networks) permite acceder de forma remota a cualquier ordenador o red situada en cualquier parte del mundo.

Grupos de noticias

Los grupos de noticias o news son grupos de discusión sobre temas muy diversos. Son listas de correo mantenidas por la red USENET. La idea

subyacente es disponer de un tablón de anuncios donde se pueden leer los mensajes de otras personas o dejar los nuestros.

Dado el gran número de grupos, se clasifican por temas y grupos, así las news son un conjunto de Newsgroups distribuidos electrónicamente en todo el mundo.

Dentro de un grupo puede haber un moderador que decide los mensajes que aparecerán en el grupo.

Listas de correo

Las listas de correo o mailing lists establecen foros de discusión a través del correo electrónico. Sólo reciben la información las direcciones de correo electrónico que se hayan subscrito. Las listas pueden ser abiertas (todo el mundo puede apuntarse) o cerradas (hay un dueño que decide quién puede pertenecer y quién no).

IRC

El IRC (Internet Relay Chat) es una conversación multiusuario que permite intercambiar mensajes por escrito.

4.3.5 Asignación y gestión de dominios

Cada ordenador de la red queda identificado por un número de 4 bytes dividido en campos de 8 bits, asignado en el protocolo IP por el NIC (Network Information Center). En la práctica es mucho más sencillo recordar direcciones alfanuméricas:

www.isa.cie.uva.es

que su correspondiente dirección numérica: 157.88.32.189

Por lo que las direcciones IP se utilizan simultáneamente con los denominados nombres de dominio (DN: Domain Name). La relación existente entre la dirección IP y el DN se gestiona mediante el DNS (Domain Name System). Cada servidor DNS es responsable de uno o más dominios. Estos servidores constituyen una red jerárquica distribuida y se encargan de almacenar y traducir los DN a las correspondientes direcciones IP.

Cada ordenador conectado a Internet tiene un número IP exclusivo. El IANA (Internet Assigned Numbers Authority) coordina este sistema asignando bloques de direcciones numéricas a registros nacionales de IP. Por ejemplo, RIPE en Europa. Entonces, los proveedores de Internet grandes tramitan la solicitud a los registros nacionales de IP para bloques de direcciones de IP. A su vez, estos proveedores reasignan direcciones a proveedores más pequeños y a usuarios finales.

Los nombres de dominio se estructuran como una jerarquía. Se dividen en dominios de primer nivel (TLDs: Top Level Domains) y éstos a su vez en dominios de segundo nivel (SLDs: Second Level Domains) y así sucesivamente. Por ejemplo, la terminación *.es* es un TLD como también lo es *.uk* o *fr*. Así podemos saber que la dirección www.uva.es corresponde a una dirección de España. Hay un pequeño grupo de dominios que no llevan un identificador nacional pero sirven para identificar la función o tarea que se desarrolla en ese dominio. La mayoría de estos dominios están asignados a ordenadores estadounidenses. Los más habituales son:

- com: organizaciones comerciales. Por ejemplo: www.amazon.com (librería americana), www.kelkoo.com (guía de compras por Internet), ...
- edu: universidades y centros educativos (en su mayoría norteamericanos). Por ejemplo: www.harvard.edu (Universidad de Harvard, USA).
- gov: organismos gubernamentales. Por ejemplo: www.nasa.gov es la dirección de la Nasa.
- mil: organismos militares. Por ejemplo: www.navy.mil (Marina americana).
- org: otras organizaciones. Por ejemplo: www.unicef.org es la página de Unicef. www.ieee.org es la página de la asociación IEEE (Institute of Electrical and Electronics Engineers), ...

El crecimiento de Internet ha sido tan espectacular que estos TLDs genéricos son pocos: se hace difícil encontrar nombres que no estén ocupados por otros y que reflejen lo que se hace en ese lugar. Así, recientemente, la ICANN (Internet Corporation for Assigned Names and Numbers) ha propuesto siete nuevos dominios:

- .aero: industria de transporte aéreo
- .biz: negocios
- .coop: cooperativas sin ánimo de lucro
- .info: uso no restringido
- .museum: museos
- .name: registro para individuos (páginas personales)
- .pro: profesiones liberales (contables, abogados, médicos)

4.4 Para saber más

[Carne] E. Bryan Carne, "Telecommunications Primer: data, voice and video communications, 2ed.", Prentice Hall PTR

[Dunlop] J.Dunlop y D. G. Smith, "Telecommunications Engineering 3ed.", Chapman&Hall

[Gitlin] Richard D. Gitlin, Jeremiah F.Hayes y Stephen B. Weinstein, "Data Communications Principles", Plenum Press

[Huidobro] Huidobro Moya, J.M., "Redes y Servicios de Telecomunicaciones", Paraninfo

[Stallings] Stallings, William, “Data and Computer Communications”, Prentice Hall

[Tanenbaum] Tanembaum, A., "Redes de Ordenadores 3e", Prentice Hall

Además de explicar los conceptos fundamentales de Internet, se ha utilizado la red de redes para encontrar información sobre algunos de los contenidos. Las direcciones que nos han resultado más útiles han sido:

http://www.khainata.com/extrainternet/inte1.html

http://internet.informaticos.org

PARTE II

5. Algorítmica

5.1 Concepto de algoritmo

La resolución de un problema científico requiere la realización de un conjunto de operaciones a partir de unos datos de entrada de modo que como resultado se obtengan un conjunto de datos de salida. Así, de un modo formal se define

> Algoritmo: "secuencia finita de operaciones que resuelve un problema en un tiempo finito"

Los principales atributos que tiene que presentar un algoritmo son que debe ser *finito*, *definido*, *preciso* e *independiente* de lenguaje de programación.

- *Finito* significa que debe tener un número finito de pasos y por tanto estar limitado tanto por el tiempo de realización como por el número de operaciones que realiza.
- *Definido* implica que ante los mismos datos de entrada se obtienen los mismos datos de salida.
- *Preciso* requiere que se indique de una forma inequívoca el orden de realización de cada paso.
- *Independiente de lenguaje de programación significa* que debe ser de propósito general, y al no contener ningún paso específico de un lenguaje en particular permitir su implementación en cualquier lenguaje de programación. Este atributo marca la clara diferencia entre algoritmo y programa de ordenador, y es que el algoritmo siempre es anterior al programa de ordenador, un algoritmo se puede codificar con diversos lenguajes de programación, dando lugar a diferentes programas de ordenador.

5.2 Elementos de un algoritmo

Un algoritmo maneja la información que se le suministra para generar una serie de resultados, así consta de *datos* y de *sentencias* para manejar esos datos. Los datos se almacenan como variables o constantes y se involucran en expresiones. Las sentencias describen las acciones algorítmicas que pueden ser

ejecutadas y en general realizan asignaciones, entradas/salidas de datos y control del flujo del algoritmo.

5.2.1 Variables, constantes y expresiones

Se define variable como

> Variable: "elemento del algoritmo que posee un valor, es conocido por un nombre o identificador y pertenece a un tipo de dato declarado al principio del algoritmo"

Antes de usar una variable debe ser *declarada*, la declaración de una variable en un algoritmo consta de una sentencia en la que se especifica el tipo de dato de la variable que se está declarando, su nombre y en ocasiones se le asigna un valor inicial.

Ejemplo: VARIABLES: *numero=10: entero, texto: string*.

En esta declaración se dice que la variable *numero* es de tipo entero y toma el valor 10 y la variable *texto* es de tipo cadena de caracteres y no tiene ningún valor asociado inicialmente.

Se define constante como

> Constante: "elemento del algoritmo similar a una variable, pero cuyo valor no cambia a lo largo del algoritmo"

y que consecuentemente debe ser inicializada de acuerdo con el tipo de dato al que pertenecen. Un concepto asociado es el de *valor constante* o *constante literal* que aparecen de forma explícita en el algoritmo y que es

> Valor constante o constante literal: "valor de cualquier tipo que se utiliza como tal"

Ejemplo: 5.38, "s", verdadero, 4.

Se define expresión como

> Expresión: "combinación de variables, constantes, valores constantes, operadores y funciones que al evaluarla en el orden correcto tiene un valor concreto"

Los tipos de expresiones más representativas son las numéricas y las lógicas.

Ejemplo: en la expresión 2*PI*r, que permite calcular la longitud de una circunferencia de radio r, se tiene que PI es una constante que tiene un valor previamente definido (3.1416), 2 es un valor constante y r es una variable que puede tomar diversos valores según sea el radio de la circunferencia considerada.

Las expresiones numéricas involucran y ofrecen como resultado datos numéricos. Los operadores **aritméticos** que involucran, según su orden de precedencia son (de mayor a menor):

1. Potenciación: ^
2. Signo: + , -
3. Producto/división: *, /
4. Suma/resta: + , -
5. Cociente/resto de división entre enteros: DIV, MOD. El operador DIV, división entera, devuelve el entero más próximo por defecto que resulta de dividir un número *a* entre otro *b*. El operador MOD devuelve el resto de la división entera de *a* entre *b*. **Ejemplo:** 5 DIV 2 ⇒2 y 5 MOD 2 ⇒1.

El orden de precedencia de los operadores, que depende del lenguaje de programación, indica en qué orden se realizan las operaciones. Sin embargo, cuando se desee forzar la evaluación de una expresión en un determinado orden, independientemente del orden de precedencia de los operadores, se utilizarán los paréntesis. Así, las reglas para evaluar una expresión son:

- Las operaciones encerradas entre paréntesis se evalúan primero. Si existen diferentes paréntesis anidados las expresiones internas se evalúan antes.
- Las operaciones aritméticas dentro de una expresión se evalúan según el orden de prioridad expresado en el apartado anterior. Si coinciden varios operadores de igual prioridad, el orden a seguir es de izquierda a derecha.

Ejemplo: sean A=1, B=2 y C=3

3*A+7⇒10

3*(A+7) ⇒24

4+C-B*2⇒3

Las expresiones lógicas o booleanas son las que ofrecen como resultado después de su evaluación un valor lógico (verdadero o falso). Involucran operadores lógicos (AND, OR, NOT) y relacionales (<,>,=,<=,>=,<>). Siendo el orden de precedencia el siguiente:

1. NOT
2. <, >, <=, >=
3. =, <>
4. AND

5. OR

Los paréntesis juegan el mismo papel que en las expresiones numéricas.

Ejemplo: sean A=1, B=2, C=3, X=VERDADERO (.V.) e Y=FALSO (.F.).

A<B⇒.V.

NOT(A<B) ⇒.F.

NOT X AND A>=C OR B<>C OR NOT (X OR Y) ⇒.F. AND .F. OR .V. OR .F.⇒.F. OR .V. OR .F. ⇒.V. OR .F. ⇒.V.

En caso de que en la expresión aparezcan tanto operadores relacionales y/o lógicos como aritméticos, estos últimos tienen un orden de precedencia menor que el NOT pero mayor que los relacionales; por ejemplo, en la expresión

A*3>C⇒.F.

primero se evalúa el producto y luego se realiza la comparación para devolver falso en caso de usar los valores de las variables definidos anteriormente.

En estas definiciones se maneja el concepto de tipo de dato que está intrínsecamente asociado a los tipos de operaciones que se pueden realizar así como al tipo de información manejada por variables y constantes. En general, como se menciona al principio de este punto, los lenguajes de programación requieren que los programas contengan una parte en la que se declaran las variables que manejan, lo cual requiere una clara definición del tipo de datos de dichas variables, por esta razón en el punto 5.3. se van a describir en detalle los tipos de datos más comunes.

5.2.2 Sentencias

Como se mencionó anteriormente las sentencias describen lo que debe hacer el algoritmo. Existen tres tipos diferentes de sentencias: de asignación, de entrada/salida y de control del flujo del algoritmo.

Sentencias de asignación

Las sentencias de asignación almacenan un valor en una variable o constante, la operación de asignación se muestra con el símbolo '←', denotando que el valor situado a su derecha se almacena en la variable o constante situada a su izquierda. Esta operación es destructiva e implica un movimiento de datos, perdiéndose el valor asignado anteriormente.

Sentencias de entrada/salida

Los datos se pueden almacenar de tres formas diferentes: asociados con constantes, asignados a una variable con una sentencia de asignación o una sentencia de lectura. Este último método, la sentencia de lectura, es el más

adecuado si se pretende realizar un programa que permita, cuando se codifique el algoritmo, manipular diferentes datos cada vez que se ejecute el programa de ordenador. La operación de lectura o de entrada permite introducir los datos desde dispositivos externos (teclado) o desde ficheros externos (unidad de disco).

Además de la sentencia de entrada existe la sentencia de salida o de escritura, tan necesaria como la de entrada, ya que un algoritmo realiza una serie de cálculos con el objetivo de obtener unos resultados, que cuando se codifique y se ejecute el consiguiente programa, podrán visualizarse mediante algún dispositivo externo (impresora, pantalla), almacenar en algún fichero externo o en un puerto de E/S (tarjeta de adquisición de datos).

5.2.3 Sentencias de control del flujo del algoritmo

Todo algoritmo puede ser escrito usando tres estructuras de control básicas: secuenciales, selectivas y repetitivas o cíclicas.

Las estructuras de control *secuenciales* son aquellas en las que todas las instrucciones o sentencias se ejecutan una después de la otra. Empezando por el inicio del algoritmo y terminando por su final.

Las estructuras de control *selectivas*, también denominadas de decisión o alternativas, se utilizan para adoptar decisiones lógicas. En este tipo de estructuras se evalúa una expresión lógica o relacional, y en función de su resultado se selecciona cual de las posibles opciones se toma. Esta estructura permite codificar bifurcaciones en el cuerpo del algoritmo así como salidas múltiples.

Las estructuras de control *repetitivas*, también llamadas cíclicas, bucles o lazos, se utilizan para realizar varias veces el mismo conjunto de operaciones. Entre ellas se encuentran aquellas en las que el número de repeticiones depende de una condición lógica o relacional (bucles controlados por bandera) y las que se manejan por un contador.

Es importante reseñar que en un mismo algoritmo pueden aparecer las tres estructuras combinadas de distintas formas y en distintas partes del mismo.

En el epígrafe destinado a describir el pseudocódigo (5.4.1) se describen los detalles de cada una de las estructuras de control.

5.3 Tipos de datos

Dado que la realización de un algoritmo es un paso previo a la codificación de un programa de ordenador y que para que la información sea procesada en un ordenador de un modo óptimo, permitiendo un almacenamiento eficiente y un acceso rápido a la información, debe estructurarse en forma de datos, se van a

estudiar los tipos de datos que maneja un ordenador y que por tanto pueden aparecer en un algoritmo.

Así, en el entorno de los lenguajes de programación se define

> Dato: "una información relativa a un objeto que es manipulable por el ordenador, que posee un valor y que es conocido en un programa o algoritmo por un nombre o identificador del dato"

Dicho identificador es un artificio para indicar una dirección de memoria.

En virtud de que el valor de un dato pueda o no cambiar durante la ejecución de un programa pueden clasificarse en *variables* o *constantes*. Así se define *dato constante* a un valor que no puede cambiar durante la ejecución de un programa, recibiendo dicho valor en el momento de la compilación. Del mismo modo se define *dato variable* aquel cuyo valor puede cambiar durante la ejecución de un programa, produciéndose dicho cambio en sentencias ejecutables.

Todos los datos tienen un tipo asociado, ya sea por definición o de forma intrínseca (como sucede a las constantes literales). El tipo de dato determina el rango de valores que puede tomar una variable y su representación interna, afectando esto último al espacio en memoria ocupado por dicho dato. Además la asignación del tipo de dato es algo necesario que permite detectar errores de operación y cómo ejecutar las operaciones (no es lo mismo 2+2 que "s"+"i", en el primer caso el resultado es 4 y en el segundo "si").

Los dos grandes tipos de datos que pueden encontrarse son *elementales* o *estructurados*. Los datos elementales o básicos son aquellos indivisibles en unidades más simples. Por el contrario, los datos estructurados son los constituidos por una agrupación lógica de elementos individuales (dato elemental u otra estructura de dato). De forma resumida, se pueden estructurar y clasificar los diferentes tipos de datos según la Figura 5-1.

5.3.1 Datos elementales

Los tipos de datos elementales dependen del lenguaje de programación usado, pero generalmente se clasifican en entero, real, lógico, carácter, enumerado, subrango y punteros.

Tipo entero

Representan número positivos o negativos sin decimales. Almacenan un valor comprendido en el siguiente rango $[-2^{n-1}, 2^{n-1}-1]$. **Ejemplo:** si n=16 el rango será [-32768,32767].

Tipo real

Almacenan un valor de la forma $N=M*B^E$. Siendo M la mantisa, B la base y E el exponente. El tamaño de la mantisa y del exponente dependen del subtipo de dato real. La precisión depende del tamaño de la mantisa, pudiendo aparecer errores de redondeo.

Tipo carácter

Representan elementos individuales de un conjunto finito y ordenado de caracteres. Existen distintos conjuntos de caracteres (ASCII, UNICODE, …), para acceder a cada uno de sus elementos puede usarse un dato de tipo entero, existiendo una relación biunívoca entre el tipo de dato carácter y el tipo de dato entero. Los lenguajes de programación disponen de funciones para acceder al carácter asociado a un determinado entero dentro de un conjunto de caracteres y viceversa.

Tipo lógico o booleano

Este tipo de dato puede tener dos posibles valores: **verdadero** o **falso**. Sobre los datos lógicos actúan operadores lógicos (AND, OR, NOR, …) y además pueden ser el resultado de una operación lógica o de relación.

Tipo enumerado

Es un tipo de datos especial que requiere que el programador defina el rango de valores que puede tomar un dato de ese tipo. El ordenador trata este tipo de valores a nivel software almacenándolos como tipo entero. Un tipo de dato enumerado podría ser *color*, pudiendo tomar los valores rojo, verde, azul, amarillo, blanco y verde.

Tipo subrango

Este tipo de dato se define a partir del tipo de dato entero, carácter o enumerado, sin más que decir que el tipo de dato definido podrá tomar un conjunto de valores limitados del tipo original.

Tipo puntero

Es aquel cuyo valor es la dirección en memoria de otro dato. Resulta especialmente útil para construir estructuras de datos.

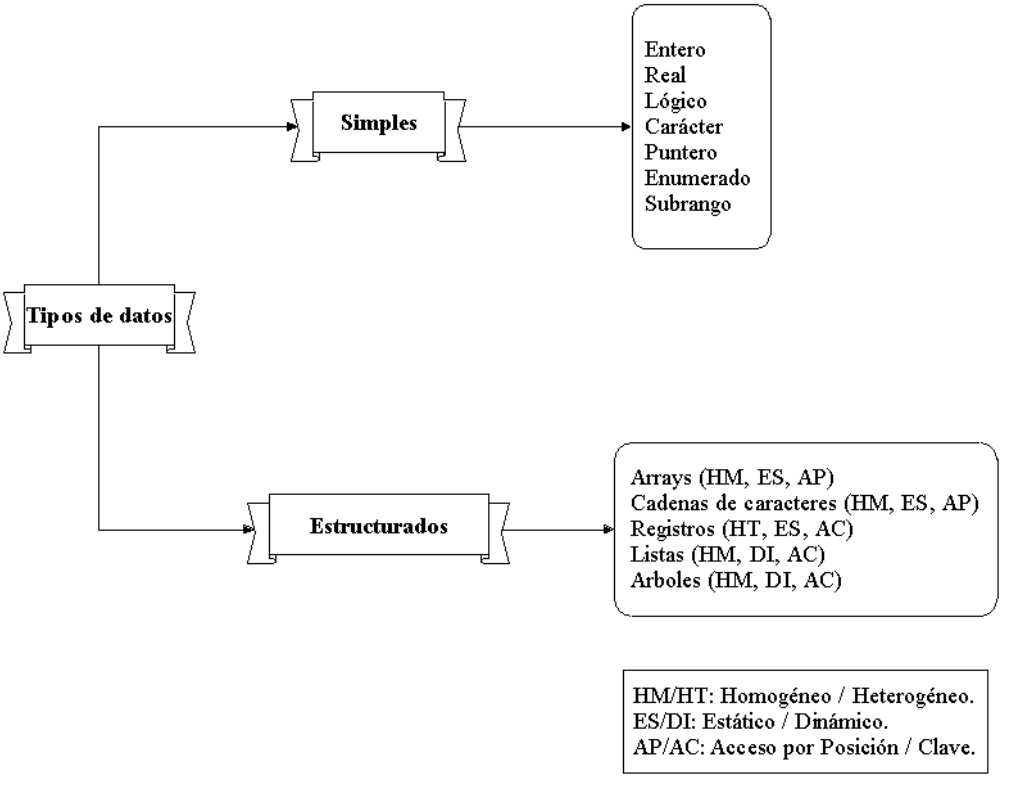


Figura 5-1. Tipos de datos

5.3.2 Datos estructurados

Previamente a la enumeración y descripción de los tipos de datos estructurados es necesario conocer los conceptos de estructura estática y dinámica y conjunto ordenado y homogéneo.

Se dice que una estructura es estática cuando el tamaño en memoria ocupado se define antes de la ejecución del programa y no puede modificarse durante la ejecución. Por el contrario una estructura dinámica es aquella en la que no se define a priori su tamaño en memoria.

Se entiende por conjunto homogéneo aquel formado por datos del mismo tipo, y ordenado si se puede acceder a cada uno de sus elementos usando un identificador.

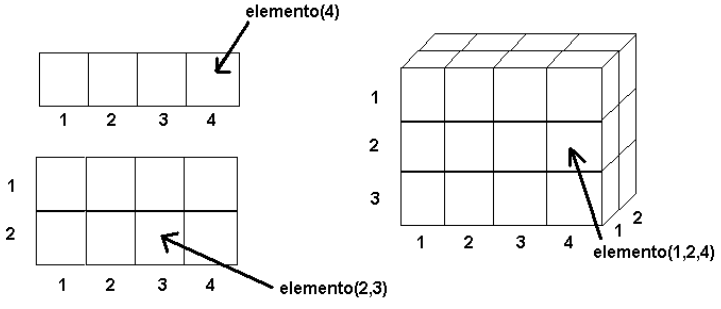
Los datos estructurados o estructuras de datos, típicamente, se clasifican en arrays, cadenas de caracteres, registros, listas y árboles.

Tipo array

Es una estructura de datos homogénea, estática y ordenada, ya que está formada por una cantidad fija de datos de un mismo tipo, cada uno de los

cuales tiene asociado uno o más índices que determinan de forma unívoca la posición del dato en el array.

Aunque no existe límite para el número de dimensiones se desaconseja superar las tres dimensiones ya que se vuelve complejo su manejo. El tipo más simple es el array unidimensional o vector y los arrays de dimensión dos se denominan tablas.



Las operaciones típicas que se pueden realizar con los arrays son: asignación, lectura/escritura, acceso secuencial, ordenación y búsqueda.

Tipo cadena de caracteres

Es una estructura de datos formada por una secuencia de caracteres en un orden determinado, siendo por tanto una estructura homogénea, estática y de acceso por posición. **Ejemplo:** frase="La puerta estaba abierta"; frase(7)="r".

Tipo registro

Es una estructura de datos formada por varios elementos o campos (simples o estructurados) que se refieren a una misma entidad. Es por tanto una estructura heterogénea, estática y de acceso por nombre. Un ejemplo típico de un tipo de dato registro sería una estructura en la que se almacenasen los datos de un alumno, siendo campos típicos el nombre, apellidos, domicilio, teléfono, fecha y lugar de nacimiento y calificaciones.

Tipo lista

Es una estructura de datos homogénea, dinámica y de acceso por clave. Están constituidos por una cantidad no prefijada de registros, con al menos dos campos, de modo que uno de ellos (puntero) sirve para localizar el siguiente elemento de la lista. Las listas son estructuras dinámicas en las que se pueden añadir, eliminar y acceder a elementos, así como comprobar si la lista está vacía o no. Existen listas tipo LIFO (*Last Input First Output*, último en entrar primero en salir) o FIFO (*First Input First Output*, primero en entrar primero en salir) según sea el modo de introducir y sacar elementos de la lista.

Tipo árbol

Es una estructura de datos homogénea y dinámica que ordena los elementos que la integran en forma de árbol, usando nodos y subárboles.

Se distinguen cuatro tipos de nodos: raíz, hoja, padre e hijo. El grado de un nodo es el número de hijos que tiene (número de nodos que directamente cuelgan de él), siendo el orden de un árbol el mayor grado de cada uno de los nodos que contiene.

Los árboles binarios son los árboles más simples y se caracterizan porque cada nodo tiene dos nodos hijos, así cada nodo será un elemento con al menos tres campos (campo llave, puntero al hijo izquierdo y puntero al hijo derecho).

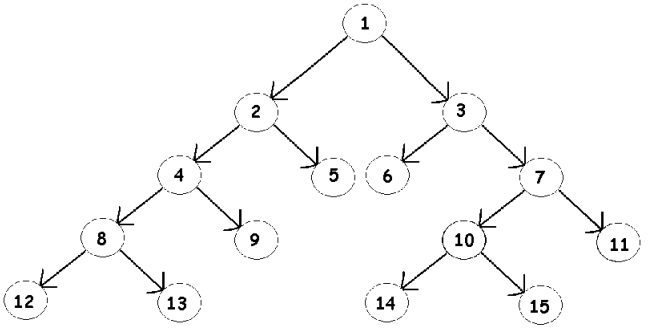


Figura 5-2 Árbol binario

La Figura 5-2 muestra un árbol binario, en el que el nodo 1 es el nodo raíz, los nodos 5, 6, 9, 11, 12, 13, 14 y 15 son nodos hoja. Además existen nodos que simultáneamente pueden ser nodo padre o hijo, como por ejemplo el nodo 2.

Las estructuras de tipo árbol son muy eficientes cuando se usan en aplicaciones en las que la búsqueda de información es una actividad básica.

5.4 Métodos de representación de algoritmos

Antes de codificar un algoritmo, una práctica recomendable, en muchas ocasiones imprescindible, es representar de un modo sencillo las operaciones que debe realizar. Existen dos tipos de métodos para representar algoritmos: métodos formales e informales.

Los métodos informales, cuyo ejemplo más claro es el lenguaje natural, tratan de describir un algoritmo como si se estuviese contando una historia. Tiene la ventaja de que es un método muy intuitivo y de fácil comprensión por cualquiera. Sin embargo, tiene la gran desventaja de que el lenguaje natural es poco adecuado para la descripción de operaciones matemáticas, siendo una descripción imprecisa que no garantiza la correcta definición del algoritmo.

Los métodos formales disponen de la sintaxis y elementos gramaticales adecuados para describir la implementación del algoritmo de resolución de un

problema en un ordenador. Existen dos tipos de métodos formales, una representación textual (pseudocódigo) o una representación gráfica (diagramas). Entre los diagramas destacan los organigramas o diagramas de flujo y los diagramas N-S (Nassi-Scheneiderman).

5.4.1 Pseudocódigo

Es un lenguaje específico de descripción de algoritmos. Puede considerarse un lenguaje natural limitado y sujeto a ciertas reglas de modo que evita la ambigüedad en la descripción del algoritmo, siendo por tanto un método preciso. Su estructura típica es la siguiente:

ALGORITMO *nombre del algoritmo*

ENTRADA: *descripción de los datos de entrada al algoritmo*.

SALIDA: *descripción de los datos de salida del algoritmo*.

VARIABLES: *lista de variables usadas separadas por comas*.

INICIO

Sentencias

FIN

Las palabras o símbolos claves que se usan dentro del cuerpo del algoritmo son:

- Operador asignación: ←.

  De forma compacta su formato es: **Variable ← Expresión**. Siendo **Variable** un identificador válido declarado anteriormente y **Expresión** una variable, constante, constante literal o fórmula a evaluar. Evidentemente el tipo de **Expresión** debe ser el mismo que el de **Variable**.

- Operaciones de entrada/salida.

  En los algoritmos las sentencias de entrada/salida se reflejan con la siguiente sintaxis, respectivamente: **LEER** *lista de variables separadas por comas* y **ESCRIBIR** *lista de variables, constantes, valores constantes y/o expresiones separadas por comas*.

- Estructuras de control secuencial. Consta de un conjunto de sentencias ordenadas.

  Ejemplo: Algoritmo para el cálculo de la superficie de un triángulo tomando como datos su base y su altura.

ALGORITMO *Superficie_triángulo*
ENTRADA: *la base y altura del triángulo*
SALIDA: *su superficie*
VARIABLES: *base, altura, superficie: reales*
INICIO
    ESCRIBIR "Introduce la base del triángulo: "
    LEER base
    ESCRIBIR "Introduce su altura: "
    LEER altura
    Superficie ← base*altura/2.
    ESCRIBIR "La superficie del triángulo es ", superficie
FIN

- Estructuras de control selectivas.

  Este tipo de estructuras permiten codificar alternativas dobles o múltiples, usando dos tipos de sentencias.

  Para el caso de alternativa doble se utiliza el siguiente bloque:

  SI *condición*
      Sentencia 1
      …
  SI NO
      Sentencia 2
      …
  FIN_SI

  Ejemplo: Algoritmo que dados dos números enteros imprime el mayor de los dos.

ALGORITMO *Mayor_de_dos*
ENTRADA: *dos números enteros*
SALIDA: *el mayor de los dos números*

```
VARIABLES: valor1, valor2: enteros
INICIO
    ESCRIBIR "Introduce el primer valor"
    LEER valor1
    ESCRIBIR "Introduce el segundo valor"
    LEER valor2
    SI valor1>valor2
        ESCRIBIR "El mayor es ", valor1
    SI NO
        ESCRIBIR "El mayor es ", valor2
    FIN_SI
FIN
```

En algunas ocasiones la selección es múltiple, por ejemplo imagínese que se quiere realizar un algoritmo de modo que dado un número entero entre 1 y 3 produzca como salida su expresión en caracteres (si introducimos 3 devuelve "tres"). Su solución requiere utilizar varios bloques SI anidados (un SI dentro de otro y anidado por la sentencia SI NO). La solución al problema propuesto podría ser la siguiente:

```
ALGORITMO texto_número
ENTRADA: un número entero entre 1 y 3
SALIDA: la expresión en caracteres del número de entrada
VARIABLES: numero: entero
           texto: cadena de caracteres
INICIO
    ESCRIBIR "Introduce un número entre 1 y 3: "
    LEER numero
    SI numero=1
        texto←"uno"
    SI NO
        SI numero=2
            texto←"dos"
        SI NO
            SI numero=3
                texto←"tres"
            SI NO
                texto←"ERROR"
            FIN_SI
        FIN_SI
    FIN_SI
    ESCRIBIR "El número ", numero, " en texto es ", texto
FIN
```

Esta solución es bastante complicada y poco elegante. Una opción mucho más práctica y eficiente es una estructura selectiva múltiple cuyo pseudocódigo genérico es:

```
EN CASO DE QUE expresión VALGA
        Valor 1: bloque sentencias 1
```

Valor 2: bloque sentencias 2

Valor 3: bloque sentencias 3

…

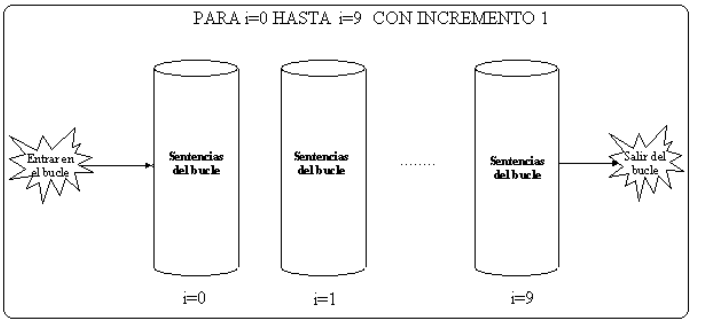
[**EN OTRO CASO** bloque sentencias x]

FIN_CASO

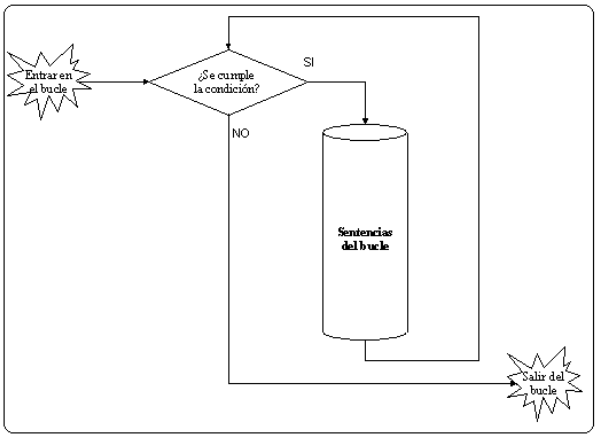
Así, el algoritmo anterior se habría codificado de este modo:

```
ALGORITMO texto_número
ENTRADA: un número entero entre 1 y 3
SALIDA: la expresión en caracteres del número de entrada
VARIABLES: numero: entero
           texto: cadena de caracteres
INICIO
    ESCRIBIR "Introduce un número entre 1 y 3: "
    LEER numero
    EN CASO DE QUE numero VALGA
        1: texto←"uno"
        2: texto←"dos"
        3: texto←"tres"
    EN OTRO CASO "ERROR"
    FIN_CASO
    ESCRIBIR "El número ", numero, " en texto es ", texto
FIN
```

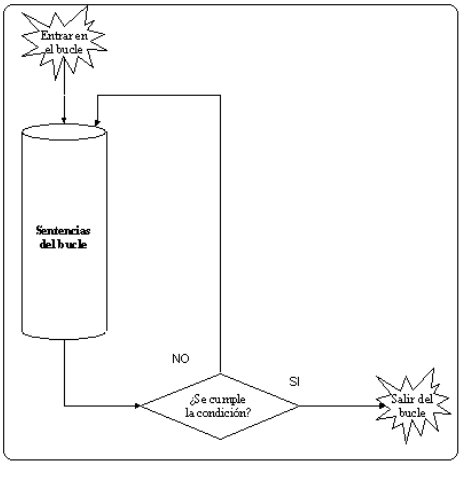
- Estructuras de control repetitivas. Como se mencionó anteriormente se utilizan cuando un bloque de sentencias debe repetirse. El número de repeticiones puede venir dado por un valor fijo o hasta que o mientras se cumpla una condición lógica. Así se tiene tres tipos de estructuras repetitivas: **PARA**, **MIENTRAS** y **REPETIR** (Figura 5-3).



BUCLE PARA



BUCLE MIENTRAS



BUCLE REPETIR

Figura 5-3

La estructura **PARA** es un bucle determinístico (ya que se conoce a priori cuantas veces debe repetirse la ejecución del bloque de sentencias) controlado por contador, repitiéndose la sentencias del bucle hasta que el contador alcanza cierto valor; su código genérico en pseudocódigo es:

PARA *contador=vi* **HASTA** *vf* **CON INCREMENTO** *n*

Bloque de sentencias

FIN_PARA

Siendo *vi* el valor inicial para la variable contador, *vf* valor para salir del bucle cuando el contador supere ese valor y *n* es el incremento del contador cada vez que se finalice la ejecución del bloque de sentencias contenidas en el bucle.

Un **ejemplo** de aplicación sería realizar el algoritmo para escribir los números pares menores o iguales que 50.

```
ALGORITMO Pares
ENTRADA: ninguna
SALIDA: los números pares menores o iguales que 50
VARIABLES: i: entero
INICIO
    PARA i=2 HASTA 50 CON INCREMENTO 2
        ESCRIBIR i
    FIN_PARA
FIN
```

Las estructuras **MIENTRAS** y **REPETIR** son ciclos no determinísticos (a priori no se conoce cuantas veces debe ejecutarse el bloque de sentencias que contiene) controlados por una condición lógica. En los bucles **MIENTRAS** la comprobación lógica se hace antes de comenzar cada ciclo, y si la evaluación de la condición lógica tiene resultado verdadero se ejecuta el contenido del ciclo y se vuelve a hacer la pregunta, si por el contrario el resultado es valor falso se sale del ciclo. En los bucles **REPETIR** la comprobación lógica se hace al finalizar cada ciclo, con lo cual al menos se ejecuta una vez, sin embargo, en los bucles **MIENTRAS** puede que no se ejecuten nunca si al entrar en el bucle la condición lógica es falsa. El pseudocódigo genérico de estas estructuras es:

MIENTRAS *condición*

Bloque de sentencias

FIN_MIENTRAS

REPETIR

Bloque de sentencias

HASTA_QUE *condición*

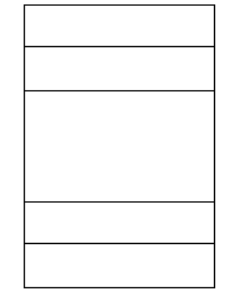
Veamos como sería el algoritmo del último problema propuesto usando las dos nuevas estructuras cíclicas.

```
ALGORITMO Pares
ENTRADA: ninguna
SALIDA: los números pares menores o iguales que 50
VARIABLES: i: entero
INICIO
  i ← 2
  MIENTRAS i ≤ 50
        ESCRIBIR i
        i ← i+ 2
  FIN_MIENTRAS
FIN
```

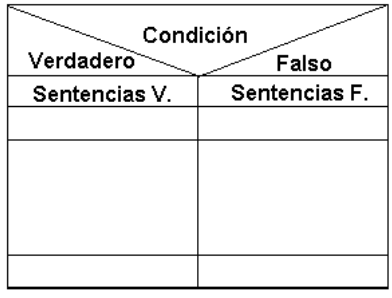
```
ALGORITMO Pares
ENTRADA: ninguna
SALIDA: los números pares menores o iguales que 50
VARIABLES: i: entero
INICIO
  i ← 0
  REPETIR
        i ← i+ 2
        ESCRIBIR i
  HASTA_QUE i = 50
FIN
```

5.4.2 Diagramas de Nassi-Scheneiderman

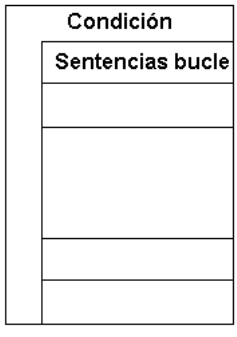
Los diagramas N-S constituyen un método gráfico de descripción de algoritmos. El símbolo básico es el rectángulo, dentro del cual se describen las acciones que constituyen el algoritmo. Su forma típica es un conjunto de cajas apiladas, dentro de las cuales pueden existir una o varias sentencias. Existen además estructuras típicas para representar las sentencias selectivas y repetitivas como se muestra en la figura adjunta.



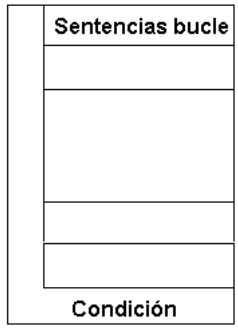
Estructura secuencial



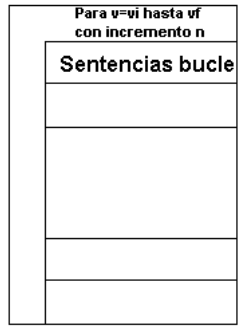
Estructura selectiva



Estructura repetitiva MIENTRAS



Estructura repetitiva REPETIR



Estructura repetitiva PARA

En este libro no se va a utilizar de forma sistemática los diagramas N-S por lo que no se hará una descripción detallada de su sintaxis. Sin embargo, para ilustrar su apariencia considérese el diagrama N-S del algoritmo que muestra los números pares menores que 50 utilizando una estructura cíclica MIENTRAS.

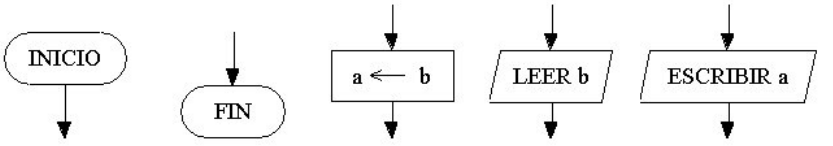
ALGORITMO *Pares*
ENTRADA: *ninguna*
SALIDA: *los números pares menores o iguales que 50*
VARIABLES: *i: entero*
INICIO
i ← 2

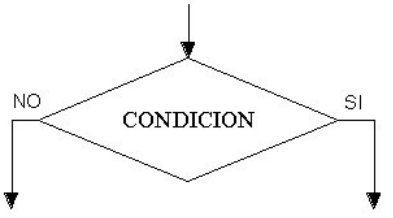
i ≤ 50	
	ESCRIBIR i
	i ← i+ 2
FIN	

5.4.3 Organigramas o diagramas de flujo

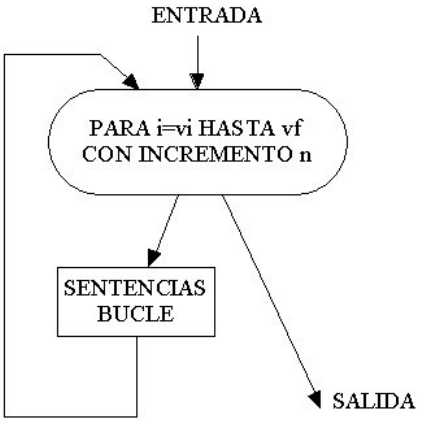
Los organigramas son otra herramienta gráfica para la descripción de algoritmos. Poseen un conjunto de símbolos finitos (figura adjunta) que permiten representar de un modo simple cualquiera de las acciones del algoritmo: inicio y fin, la sentencia de asignación, las sentencias de E/S de datos y las estructuras de control selectivas y repetitivas. Además, los símbolos están unidos por flechas, denominadas líneas de flujo, que indican el orden o secuencia de las acciones.

A modo de **ejemplo** se desarrollan los organigramas del algoritmo en el que introducidos dos números enteros muestra el mayor de ellos y las tres variantes del algoritmo en el que se muestran los números pares menores o iguales a 50.

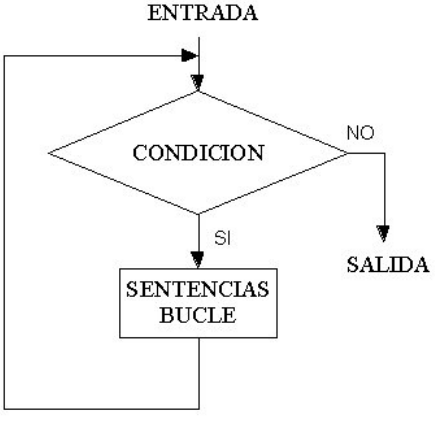




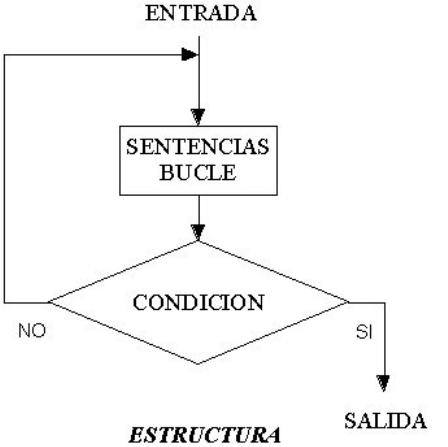
ESTRUCTURA SELECTIVA



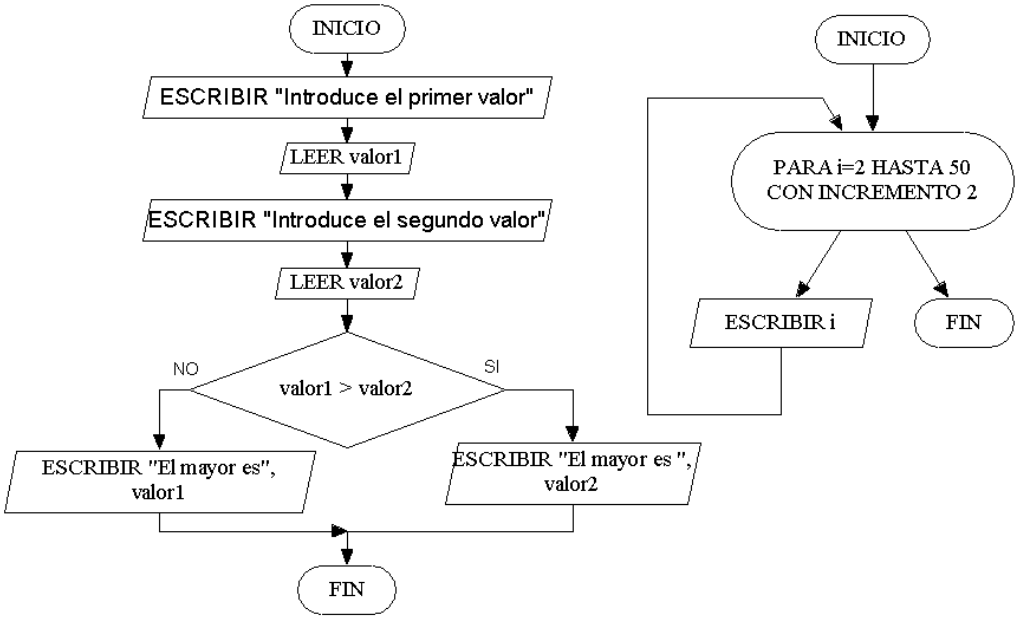
ESTRUCTURA REPETITIVA PARA

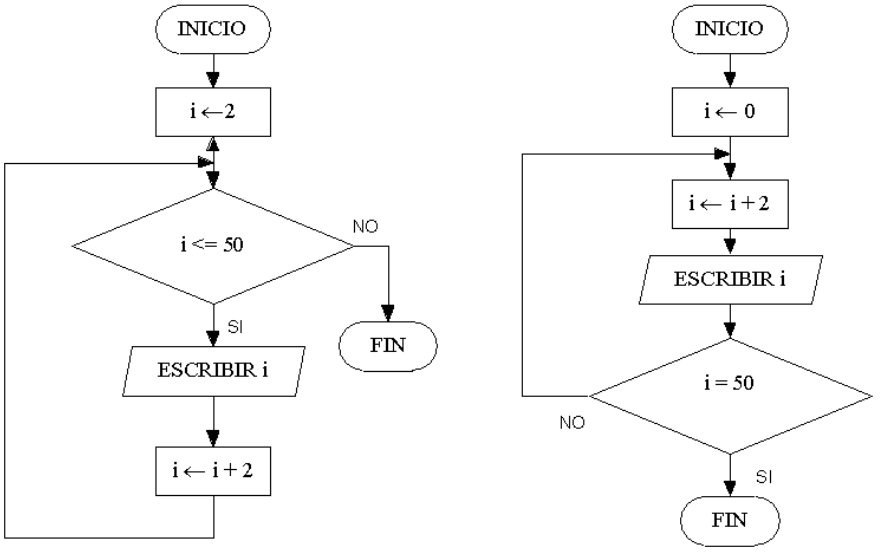


ESTRUCTURA REPETITIVA MIENTRAS



ESTRUCTURA REPETITIVA HASTA





5.5 Subalgoritmos

Un problema complejo puede dividirse en subproblemas más sencillos que permiten su solución. Esta característica de descomponibilidad puede aplicarse a los algoritmos posibilitando un diseño descendente y jerárquico de un determinado algoritmo, de modo que exista un algoritmo principal que llame a determinados subalgoritmos y estos a su vez a otros subalgoritmos, hasta

llegar a los niveles más bajos de la jerarquía donde se sitúan los subalgoritmos que resuelven problemas especiales y comunes a los demás.

Este diseño descendente de los algoritmos permite que los algoritmos sean **simples**, **modulares** y **reutilizables**, lo cual facilita la lectura, comprensión, prueba y verificación del algoritmo, así como su posterior codificación en un lenguaje de programación.

El diseño de un subalgoritmo es similar al de un algoritmo, pero como realiza una función u operaciones para otro algoritmo o subalgoritmo debe recibir datos y/o devolver resultados, debiéndose establecer una comunicación entre el subalgoritmo llamado y el algoritmo o subalgoritmo llamante. Esta comunicación se denomina paso de parámetros, existiendo dos tipos de parámetros: *formales* y *actuales*. Los **parámetros formales** son las variables utilizadas por el subalgoritmo llamado para la emisión o recepción de datos a o desde el algoritmo llamante. Los **parámetros actuales** son las variables, constantes o expresiones utilizadas por el algoritmo llamante.

El paso de parámetros puede realizarse *por valor* o *por referencia*. Cuando un parámetro se pasa por valor, no se modificará en el subalgoritmo, ya que este copia el valor en el correspondiente argumento formal para utilizarlo. Por el contrario, cuando un argumento se pasa por referencia, el algoritmo o subalgoritmo llamante le pasa al subalgoritmo llamado la dirección de la variable, así, un parámetro por referencia sí podrá ser modificado en el subalgoritmo.

Debe añadirse que existen datos *globales* que son accesibles por todos los subalgoritmos y datos *locales* que se restringen al algoritmo o subalgoritmo en el que están definidos.

Los subalgoritmos se clasifican en funciones y procedimientos. Las funciones reciben valores de entrada y devuelven un valor que es el resultado de la función (ejemplo: un subalgoritmo que calcula el producto escalar de dos vectores). Un algoritmo invoca una función utilizando un nombre y una lista de parámetros actuales. Si se desea que el subalgoritmo devuelva más de un valor se usan los procedimientos (ejemplo: un subalgoritmo que calcula la entalpía y temperatura del vapor saturado en función de su presión). Su invocación se realiza usando su nombre y una lista de parámetros actuales, tanto los de entrada como los de salida. Por supuesto cuando se define una función o un procedimiento debe establecerse una correspondencia adecuada entre parámetros formales y actuales.

Ejemplo: realice un algoritmo que dados dos vectores de 3^2, calcule su producto escalar y el coseno del ángulo que forman.

```
ALGORITMO Producto escalar y ángulo formado por dos vectores
ENTRADAS: coordenadas cartesianas de los vectores
SALIDAS: su producto escalar y el ángulo que forman
VARIABLES: xv1, yv1, xv2, yv2, vprod, vang: reales
INICIO
   ESCRIBIR "Coordenada x del vector V1: "
   LEER xv1
   ESCRIBIR "Coordenada y del vector V1: "
   LEER yv1
   ESCRIBIR "Coordenada x del vector V2: "
   LEER xv2
   ESCRIBIR "Coordenada y del vector V2: "
   LEER yv2
   PROD_COS (vprod, vang, xv1, yv1, xv2, yv2)
   ESCRIBIR "Producto escalar: ", vprod
   ESCRIBIR "Ángulo: ", vang*180/PI
FIN
```

```
PROCEDIMIENTO PROD_COS (prod, ang_rad, x1, y1, x2, y2)
ENTRADAS: x1, y1, x2, y2: reales
SALIDAS: prod, ang_rad: reales
VARIABLES: mod1, mod2: reales
INICIO
        prod ← x1*x2 + y1*y2
        mod1 ← MODULO (x1, y1)
        mod2 ← MODULO (x2, y2)
        ang_rad ← acos(prod/(mod1*mod2))
FIN
```

```
FUNCION MODULO (x, y)
ENTRADAS: x, y: reales
SALIDA: m: real
INICIO
        m ← sqrt(x^2 + y^2)
FIN
```

En el algoritmo desarrollado no existen datos globales, todos son locales. Sin embargo, se podría haber realizado de modo que las variables que almacenan los valores de las coordenadas cartesianas de los vectores dato fuesen variables globales que se hubiesen podido utilizar en el procedimiento PROD_COS sin necesidad de haber realizado el paso de parámetros entre el algoritmo principal y el susodicho procedimiento.

5.6 Recursividad

La recursividad es una técnica de programación en la que desde una función se realiza una llamada a sí misma.

El ejemplo más típico de programación recursiva es el cálculo del factorial de un número natural (el caso de número real requiere otro tipo de solución). Es conocido que el factorial de un número natural **n** (**n!**) es **n·(n-1)·(n-2)·(n-3)·...·1** y que dos de sus principales propiedades son que **0!= 1** y que **n!=n·(n-1)!**. Así, el factorial de un número natural mayor que uno es igual a sí mismo multiplicado por el factorial del número natural anterior. De modo que una función que devuelva el valor del factorial de un número natural, **n**, debe llamarse a sí misma con el parámetro de entrada **n-1**.

En programación, la recursividad no es una técnica indispensable ya que cualquier algoritmo recursivo puede implementarse de una forma iterativa. En general, un algoritmo diseñado con técnicas recursivas es más elegante, fácil de entender y requiere menos código que uno basado en técnicas iterativas, pero es en particular cada algoritmo quien decide qué tipo de solución se le ajusta mejor.

Sin embargo, la recursividad no es la panacea, también puede ser una fuente de problemas ya que es posible crear una función recursiva que nunca llegue a devolver un valor porque no tenga una condición de finalización, generándose lo que se denomina bucle "infinito". Existen unas reglas para diseñar algoritmos recursivos que si son aplicadas de forma exhaustiva conducen a algoritmos correctos. Estas reglas son:

1. Siempre han de existir una o varias condiciones lógicas en función de los argumentos de entrada para las que la función devuelve un valor sin

necesidad de llamarse a sí misma, es lo que se denomina la **solución trivial** del problema.

2. Cuando los valores de los argumentos de entrada no cumplan las condiciones lógicas que conducen a la solución trivial se llamará recursivamente a la función, es la llamada **solución no trivial** del problema.
3. Asegúrese que el valor que devuelve para el caso no trivial es la solución correcta del algoritmo recursivo. Es decir, que está correctamente programada la solución recursiva del algoritmo, este concepto es la base del denominado **principio de inducción**.

A continuación se muestra el pseudocódigo de una función que calcula el factorial de un número natural.

```
FUNCION FACTORIAL (n)
ENTRADA: n: entero
SALIDA: fact: entero
INICIO
    SI n<0
        ESCRIBIR "Número no natural"
        fact ← -1
    SI NO
        SI n=0
            fact ← 1
        SI NO
            fact ← n*FACTORIAL(n-1)
        FIN_SI
    FIN_SI
FIN
```

Como se puede observar, la función hace uso de la propiedad recursiva del factorial (**n!=n·(n-1)!)**, que es la solución no trivial y del valor del factorial de 0 (**0!= 1)**, que es la solución trivial. Nótese que los únicos valores válidos de los argumentos de entrada son números naturales.

Una forma iterativa de abordar el problema del cálculo del factorial de un número natural podría ser la siguiente:

```
FUNCION FACTORIAL (n)
ENTRADA: n: entero
SALIDA: fact: entero
VARIABLES i: entero
INICIO
        fact ← 1
        SI n<0
                ESCRIBIR "Número no natural"
                fact ← -1
        SI NO
                SI n=0
                        fact ← 1
                SI NO
                        PARA i=1 HASTA n
                                fact ← fact*i
                        FIN_PARA
                FIN_SI
        FIN_SI
FIN
```

5.7 Problemas propuestos

Desarrolle los algoritmos que resuelven los siguientes problemas:

1. Un comerciante necesita conocer el precio al que debe vender un producto, para ello, dado el coste de dicho producto le incrementa su valor en un 25% y al resultado le aplica el IVA correspondiente (en este caso el 16%). El resultado debe ser el precio final y el valor del impuesto.
2. Dados dos números reales, se necesita saber el resultado de dividir el mayor de ellos entre el cuadrado del menor.
3. Ingresar los valores de los coeficientes a, b, c de una ecuación de segundo grado ($ax^2+bx+c=0$) y obtener como resultado sus raíces. En el caso de que

sean complejas conjugadas las soluciones deben ser de la forma n+mi y n-mi siendo i la unidad imaginaria.

4. Dados dos números reales y un operador numérico (+, -, *, /) devuelva el valor de la operación `num1 operador num2`. Si se ingresa un símbolo de operador diferente a los enumerados indicar que la operación solicitada no está prevista.
5. Ingresar diez números enteros y devolver la suma de los valores positivos (ignorar los nulos).
6. Dado un número entero del 1 al 10, devolver su tabla de multiplicar del 1 al 10.
7. Introducir una serie de números reales, indicando el fin de la serie con el valor nulo, y calcular su media aritmética.
8. Considérese una empresa comercial con veinte vendedores, cada uno identificado con un código del 1 al 20, y conocido el importe mensual de las ventas de cada uno de ellos. Calcúlese el importe total de las ventas del mes, y el número de vendedores que superaron el importe mensual medio de ventas. NOTA: utilícense estructuras de dato tipo array.
9. Desarrolle una función que de un modo recursivo encuentre el cero de una función real de variable real continua tomando como dato los extremos de un intervalo que contiene al cero de la función y en cuyos extremos la función cambia de signo (ayuda: utilice el método de la bisección del intervalo que contiene al cero).

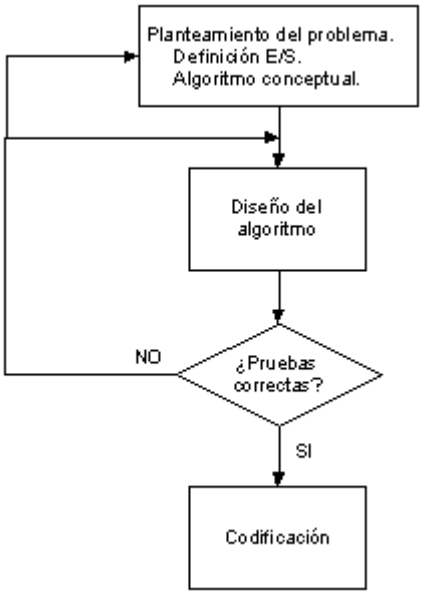
6. Resolución de problemas

6.1 Metodología de resolución

La resolución de problemas científicos y técnicos con ordenador requiere la codificación de un programa de ordenador que solicite unos datos y que después de realizar un secuencia de operaciones devuelva un conjunto de resultados. Evidentemente, cuando los problemas sean complejos será necesario un análisis y diseño previo a la codificación del programa de ordenador. Los pasos que deben seguirse en la fase de análisis y diseño son:

- **Planteamiento del problema**, decidiendo los datos de entrada y los resultados o datos de salida que se pretenden obtener y realizando un algoritmo conceptual que resuelva el problema. Este algoritmo conceptual debe ser básico y genérico, incluyendo las operaciones que deben realizarse con los datos. En ocasiones es conveniente pensar en un ejemplo sencillo del problema que se quiere resolver y aplicar el algoritmo conceptual.

- Usando el algoritmo conceptual se **diseña el algoritmo** que resuelve el problema, usando para su descripción pseudocódigo o, preferentemente, algún diagrama de flujo. El diseño del algoritmo debe seguir el método descendente, utilizando procedimientos y funciones cuando se consideren necesarias.



- Realizado el algoritmo, puede **probarse con algunos ejemplos** sencillos y ver si funciona bien, si en algún caso no funciona bien, se debe comprobar si el planteamiento es correcto o no y si el algoritmo está bien diseñado o no. En caso de que la fase de pruebas resulte positiva se procede a la codificación de dicho algoritmo.

6.2 Casos típicos

6.2.1 Sumatorios y medias aritméticas

El empleo de sumatorios y medias aritméticas es algo muy usual en los problemas científicos y técnicos. Así, por ejemplo, cuando se realiza un experimento de laboratorio, el resultado se obtiene al realizar una media aritmética de un conjunto de datos. También, algunos problemas científicos, como aquellos que tratan con sistemas de partículas, requieren el uso intensivo de sumatorios. Y, por supuesto, en la vida cotidiana existen multitud de situaciones que requieren el uso de sumatorios y medias; por ejemplo, los fondos de inversión referidos a índices bursátiles cuyo rendimiento es función de su evolución en un periodo de tiempo.

En programación, cuando se planteen problemas relacionados con sumatorios y medias aritméticas se usan dos tipos de variables, una se utiliza como contador y la otra como acumulador. De este modo, cuando se realiza un programa de ordenador que calcula la media aritmética de un conjunto de datos, existe una variable tipo contador en la que se va anotando el número de datos usados y otra en la que se va sumando el valor de cada uno de ellos, así, cuando se hayan considerado todos los datos, su media será el cociente de la variable acumulador entre la variable contador.

Para ilustrar el uso de sumatorios y medias se ha elegido un sencillo problema relacionado con la Física, en el que dado un conjunto de partículas, de las que son conocidas sus posiciones (en un sistema de referencia cartesiano) y sus masas, se pide calcular la posición del centro de masas del sistema de partículas así como el valor medio de la masa de las partículas.

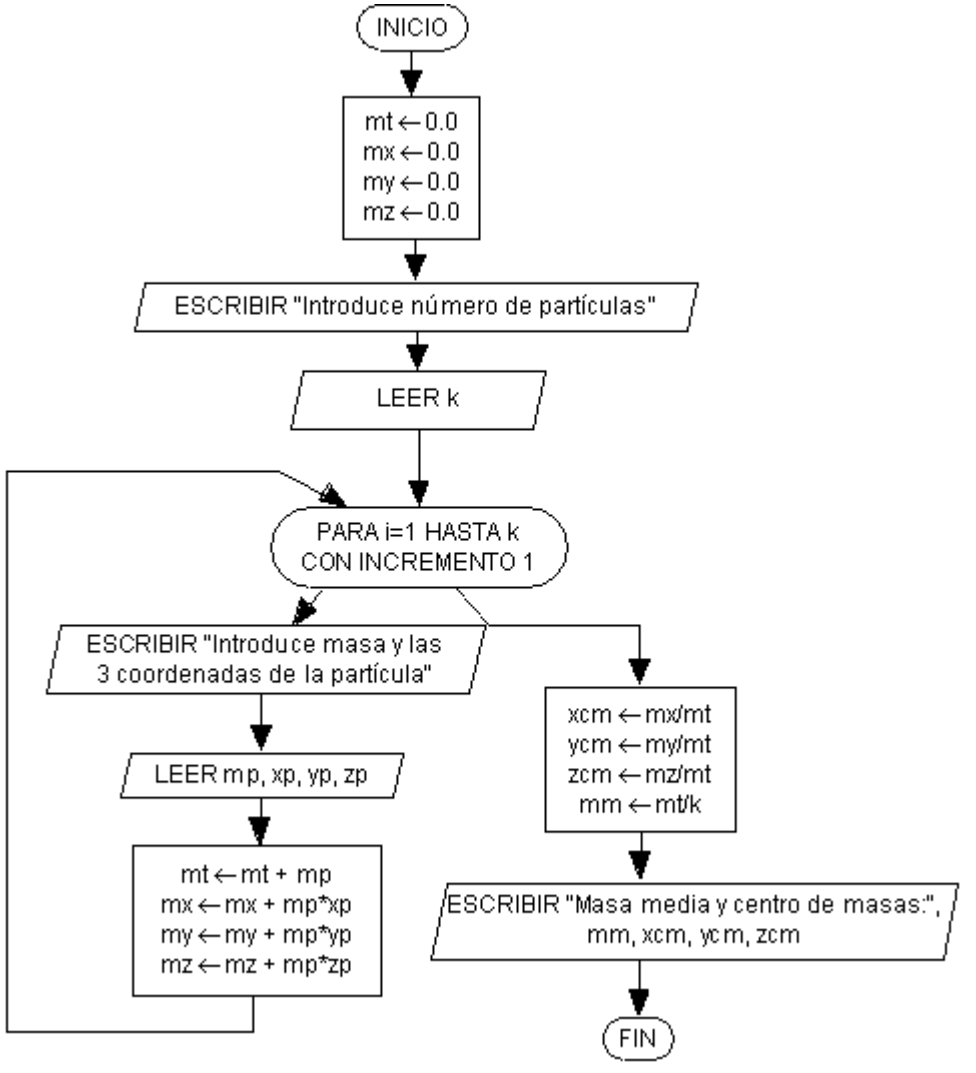
Resolución del problema

Es conocido que dado un sistema de partículas la posición del centro de masas, en coordenadas cartesianas, viene dado por las siguientes expresiones: $X_{cm} = \frac{\sum_{i=1}^{k} x_i \cdot m_i}{\sum_{i=1}^{k} m_i}; Y_{cm} = \frac{\sum_{i=1}^{k} y_i \cdot m_i}{\sum_{i=1}^{k} m_i}; Z_{cm} = \frac{\sum_{i=1}^{k} z_i \cdot m_i}{\sum_{i=1}^{k} m_i}$, siendo, respectivamente, x_i, y_i, z_i y m_i las coordenadas y la masa de la i-ésima partícula de un conjunto de k partículas, y $\overline{m} = \frac{\sum_{i=1}^{k} m_i}{k}$ es el valor medio de la masa del sistema de partículas.

Para resolver el problema se plantea un algoritmo que solicita primero el número de partículas que forman el sistema y que después, mediante un bucle

PARA, solicita, sucesivamente, los valores de las masas (*mp*) y coordenadas de cada partícula (*xp*, *yp*, *zp*) y acumula los valores de la masa total (*mt*) y de la masa en las diversas coordenadas cartesianas (*mx*, *my*, *mz*). Al finalizar el bucle calculará las coordenadas cartesianas del centro de masas del sistema (*xcm*, *ycm*, *zcm*) y el valor de la masa media de dicho sistema (*mm*).

```
ALGORITMO Centro de masas
ENTRADA Número de partículas, masa y coordenadas de cada una de ellas.
SALIDA Masa media y centro de masas
VARIABLES k, i: enteros
          mp, xp, yp, zp, mt, mx, my, mz, xcm, ycm, zcm, mm: reales
INICIO
    mt←–0.
    mx←–0.
    my←–0.
    mz←–0.
    ESCRIBIR "Introduce el número de partículas"
    LEER k
    PARA i=1 HASTA k CON INCREMENTO 1
        ESCRIBIR "Introduce masa y 3 coordenadas de la partícula: "
        LEER mp, xp, yp, zp
        mt ← mt + mp
        mx ← mx + mp*xp
        my ← my + mp*yp
        mz ← mz + mp*zp
    FIN_PARA
    xcm ← mx/mt
    ycm ← my/mt
    zcm ← mz/mt
    mm ← mt/k
    ESCRIBIR "Masa media y centro de masas: ", mm, xcm, ycm, zcm
FIN
```



Para probar la validez del algoritmo planteado se puede tratar de calcular la posición del centro de masas de un sistema de cuatro partículas de masa unidad y dispuestas en forma de cuadrado cuya longitud del lado es dos y centrado en el origen de coordenadas. Evidentemente el centro de masas está situado en el origen de coordenadas y la masa media es la unidad. Se puede añadir alguna comprobación adicional desplazando el cuadrado y modificando la masa de alguna de las partículas situadas en sus vertices.

6.2.2 Resolución de ecuaciones no lineales

El planteamiento de numerosos problemas científico-técnicos conduce a ecuaciones no lineales que deben resolverse para obtener la solución del problema. En algunos casos pueden ser resueltas de forma analítica, en un número predeterminado y finito de pasos, como por ejemplo $x^k = N$, siendo

N un número real positivo, ya que $x = 10^{\frac{1}{k}\log(N)}$. Sin embargo, en ocasiones no existen métodos analíticos para resolverlas, por lo que se recurre a expresar la ecuación de la forma $f(x) = 0$ y usar los métodos de aproximaciones sucesivas.

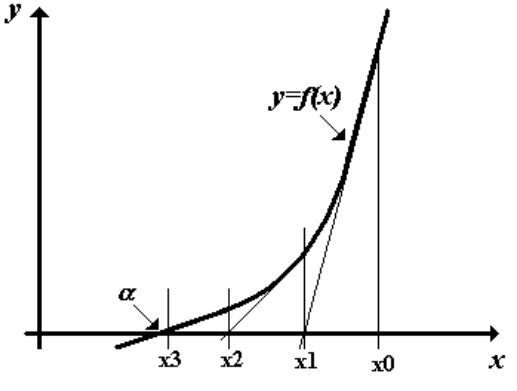
Existen diversos métodos de aproximaciones sucesivas; como este no es un libro de cálculo numérico no se hará revisión de ellos, sin embargo, se realizará el algoritmo del método de Newton-Raphson, y se tratará el problema de realizar algoritmos en los cuales se generen una secuencia de números y en los que a priori no se sabe el número de iteraciones que realizará el ordenador, aunque sí los criterios para finalizar la ejecución del programa.

Resolución del problema

A continuación se describe brevemente el método de Newton-Raphson para posteriormente realizar el organigrama de dicho algoritmo.

Este método trata de encontrar una raíz α de la ecuación $f(x) = 0$, y está basado en el desarrollo en serie de Taylor de la función $f(x)$ en torno al punto x_i, siendo x_i una estimación inicial de la raíz α.

Supóngase que se dispone de una estimación inicial, x_i, de una raíz real de la ecuación real $f(x) = 0$. La ecuación de la tangente a $f(x)$ en $x=x_i$ puede expresarse como $y(x) = f(x_i) + f'(x_i)(x - x_i)$. Sea el punto $(x_{i+1},0.)$ la intersección de la tangente con el eje x; por tanto debe cumplirse que: $0. = f(x_i) + f'(x_i)(x_{i+1} - x_i)$ y resolviendo la ecuación para x_{i+1} se tiene que $x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$, que es la fórmula clásica del método iterativo de Newton-Raphson. Sin analizar la convergencia del método puede decirse que aplicando está fórmula iterativa la solución de la ecuación $f(x) = 0$ se obtiene cuando la distancia entre x_{i+1} y x_i es menor que una determinada cota de error ε.



Este caso es un claro ejemplo de procedimientos iterativos, muy comunes en cálculo numérico. En estos procedimientos se trata de generar una sucesión de valores de una expresión a partir de un valor inicial, finalizando cuando la diferencia entre dos de ellos es menor que una cota de error o cuando se ha

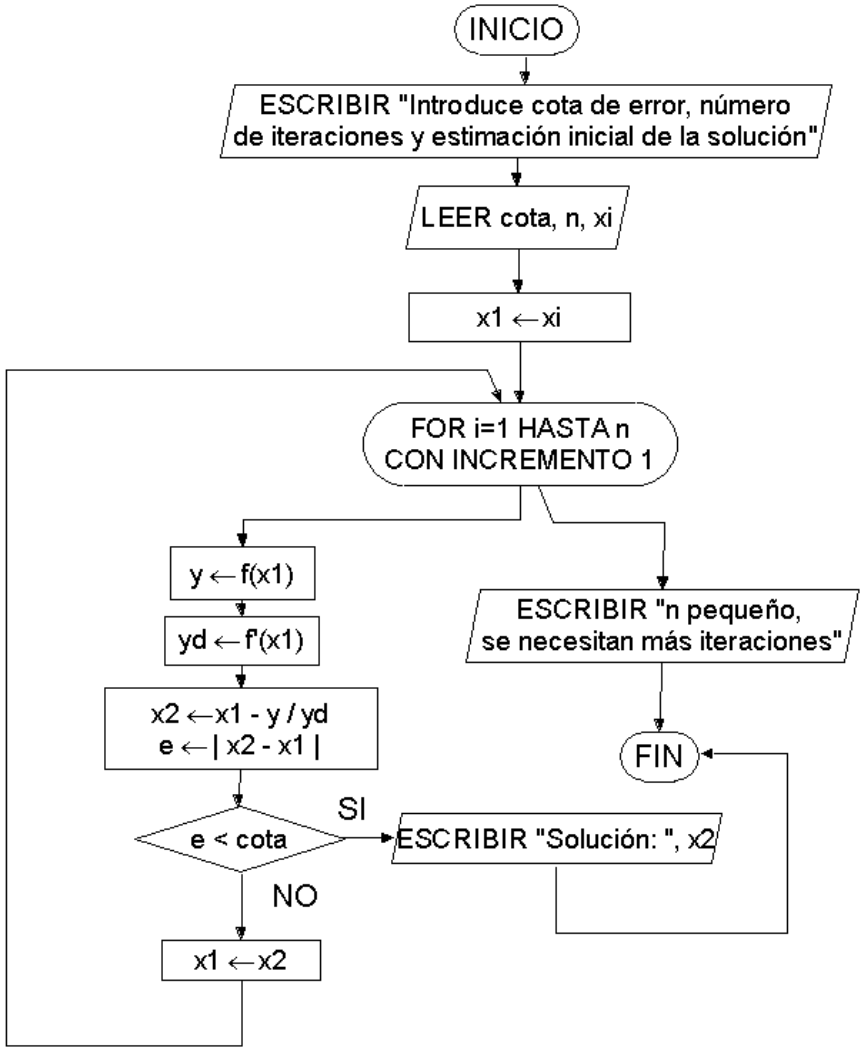
realizado un número determinado de iteraciones N sin que se haya satisfecho la condición de finalización.

Un algoritmo que resolviese una ecuación de la forma $f(x) = 0$ utilizando el método expuesto, debe solicitar inicialmente la cota de error (*cota*), el número máximo de iteraciones (*n*) y una estimación inicial de la solución (*xi*). A continuación, evaluaría la formula iterativa cuyo resultado es la estimación actual (*x2*) y calcularía la diferencia entre la estima pasada (*x1*) y la actual. En función de esa diferencia determinará si ha encontrado o no la solución. En caso de que aún no se haya satisfecho el criterio de error, y no se haya alcanzado el número máximo de iteraciones, la estimación actual pasará a ser la estimación pasada, y se incrementará el número de pasos realizados (*i*) en uno y se volverá a repetir todo el proceso.

Fíjese que para resolver este problema se necesita recurrir a estructuras repetitivas del tipo HASTA, ya que no se conoce a priori si el algoritmo converge o no, y en el caso de que así sea no se sabe en que iteración se va a producir la convergencia. Si se desea, se puede utilizar un bucle de tipo PARA asociado al número máximo de iteraciones, ya que es dato conocido al principio del algoritmo, pero en ese caso habría que *romper* el bucle en caso de que se alcanzara la solución antes de llegar al máximo de iteraciones.

```
ALGORITMO Método de Newton-Raphson
ENTRADA Cota de error, número de iteraciones y estimación inicial
SALIDA Solución de la ecuación f(x)=0
VARIABLES
    n, i: enteros
    y, yd, xi, x1, x2, cota: reales
INICIO
    ESCRIBIR "Introduce la cota de error, número de iteraciones y la
    estimación inicial de la solución"
    LEER cota, n, xi
    x1←xi
    PARA i=1 HASTA n CON INCREMENTO 1
        y ← f(x1)
        yd ← f'(x1)
        x2←x1- y/yd
        SI abs(x2-x1)<cota
            ESCRIBIR "Solucion: ",x2
```

```
                FIN
            SI_NO
                x1←x2
            FIN_SI
        FIN_PARA
        ESCRIBIR "Número pequeño de iteraciones"
FIN
```



En el capítulo relativo a la programación de algoritmos en C y MATLAB se aplicará este método para resolver la ecuación $x^2 + 0.5 - e^{-x} = 0.$. Siendo en este caso $f(x) = x^2 + 0.5 - e^{-x}$ y $f'(x) = 2 \cdot x + e^{-x}$.

6.2.3 Operaciones con vectores

Las operaciones con vectores son una herramienta habitual para plantear y resolver problemas científicos, siendo por sí mismas de importancia capital en disciplinas como el Algebra. En la Física es habitual recurrir al producto escalar entre vectores cuando se trate de resolver problemas asociados a la energía y al trabajo, el producto vectorial se usa de forma extensiva en problemas relativos a la dinámica de rotación (cálculo de momentos angulares, par de una fuerza).

Resolución del problema

En este epígrafe se plantea un algoritmo que dados dos vectores $\mathbb{R}^3$ calcula sus productos escalar y vectorial, así como el ángulo que forman. Este algoritmo tiene sentido por sí mismo, pero tiene un valor añadido, ya que si se codificase en forma de procedimiento o dividiéndole en dos (uno para el producto escalar y otro para el vectorial) permitiría hacer llamadas desde un programa principal siempre que se necesitase calcular las magnitudes enumeradas anteriormente.

De forma secuencial el algoritmo consta de los siguientes pasos:

- Inicialización de las variables que van a servir como acumuladores para calcular el producto escalar (*productoe*) y el módulo de los dos vectores (*modulov1* y *modulov2*).
- Lectura por teclado de los dos vectores (*v1* y *v2*); como son dos vectores de tres componentes se utilizan dos ciclos PARA con contador de uno a tres.
- Utiliza un bucle PARA para calcular el producto escalar y el cuadrado del módulo de cada vector mediante las siguientes fórmulas: $\bar{u}\cdot\bar{v} = u_1\cdot v_1 + u_2\cdot v_2 + u_3\cdot v_3$ y $|\bar{u}|^2 = u_1\cdot u_1 + u_2\cdot u_2 + u_3\cdot u_3$. Al salir del bucle, calcula cada una de las tres componentes que resultan del producto vectorial de los dos vectores según: $\bar{u}\wedge\bar{v} = \begin{vmatrix} \bar{i} & \bar{j} & \bar{k} \\ u_1 & u_2 & u_3 \\ v_1 & v_2 & v_3 \end{vmatrix}$, además calcula el ángulo entre los dos vectores dados usando la otra fórmula que permite el cálculo del producto escalar de dos vectores: $\bar{u}\cdot\bar{v} = |\bar{u}|\cdot|\bar{v}|\cos(\alpha)$.
- Finalmente se presentan resultados.

Nótese que en este algoritmo se usan dos funciones que están definidas en casi todos los lenguajes de programación: la raíz cuadrada positiva (SQRT) y la inversa de la función coseno (ACOS).

ALGORITMO Productos vectorial y escalar

ENTRADA Vectores dato

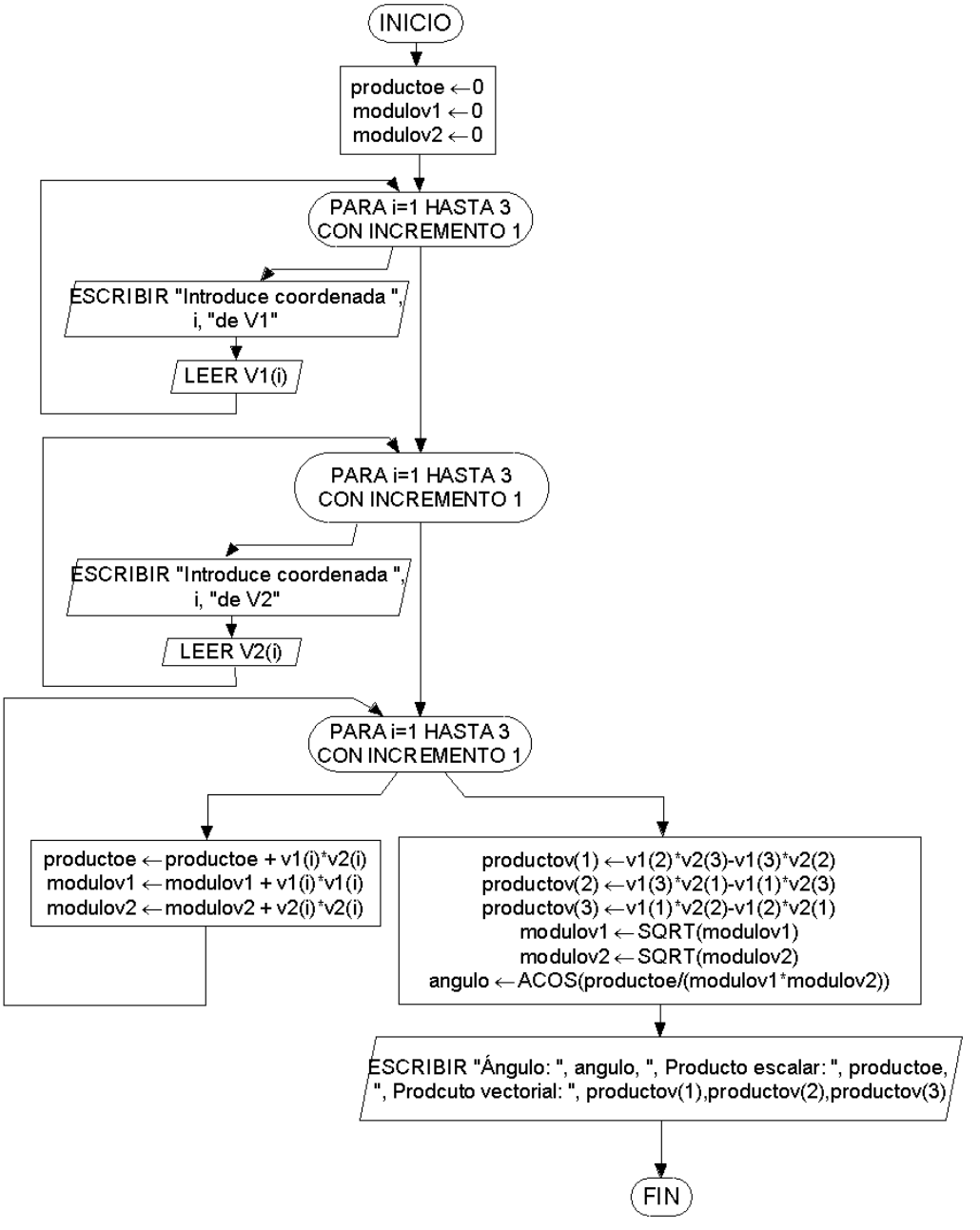
```
SALIDA Producto escalar, vectorial y el ángulo que forman
VARIABLES
        i: entero
        v1(3), v2(3), productov(3): vectores reales
        productoe, modulov1, modulov2, angulo: reales
INICIO
        productoe←0.
        modulov1←0.
        modulov2←0.

        PARA i=1 HASTA 3 CON INCREMENTO 1
                ESCRIBIR "Componente ", i, "del vector v1: "
                LEER v1(i)
        FIN_PARA

        PARA i=1 HASTA 3 CON INCREMENTO 1
                ESCRIBIR "Componente ", i, "del vector v2: "
                LEER v2(i)
        FIN_PARA

        PARA i=1 HASTA 3 CON INCREMENTO 1
                productoe← productoe+v1(i)*v2(i)
                modulov1← modulov1+v1(i)*v1(i)
                modulov2← modulov2+v2(i)*v2(i)
        FIN_PARA

        productov(1)←v1(2)*v2(3)-v1(3)*v2(2)
        productov(2)←v1(3)*v2(1)-v1(1)*v2(3)
        productov(3)←v1(1)*v2(2)-v1(2)*v2(1)
        modulov1←sqrt(modulov1)
```

modulov2←sqrt(modulov2)

angulo←acos(productoe/(modulov1*modulov2))

ESCRIBIR “Ángulo: “, angulo, “, Producto escalar: “, productoe, “, Producto vectorial: “, productov(1), productov(2), productov(3)

FIN



6.2.4 Resolución de sistemas de ecuaciones lineales

En casi todas las disciplinas científicas existen problemas cuya solución requiere resolver un sistema de ecuaciones lineales de la forma:

$$a_{11} \cdot x_1 + a_{12} \cdot x_2 + \cdots + a_{1n} \cdot x_n = b_1$$
$$a_{21} \cdot x_1 + a_{22} \cdot x_2 + \cdots + a_{2n} \cdot x_n = b_2$$
$$\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots$$
$$a_{n1} \cdot x_1 + a_{n2} \cdot x_2 + \cdots + a_{nn} \cdot x_n = b_n$$

Por ejemplo, una red eléctrica de componentes resistivos puede expresarse de esta forma, donde los coeficientes a_{ij} son números reales que representan los valores de las resistencias, las incógnitas x_i representan los valores de las corrientes y las constantes reales b_j representan los valores de las fuentes de voltaje.

Otro caso típico es cuando se reemplazan ecuaciones diferenciales en parciales por ecuaciones lineales en diferencias y se resuelven por métodos de álgebra lineal.

Existe una teoría bien establecida para la resolución de sistemas de ecuaciones lineales, en este libro ilustraremos cómo utilizar la regla de Cramer para resolver sistemas lineales de tres ecuaciones. Ya que, aunque este método es de aplicación general para sistemas de n ecuaciones, no es apropiado cuando el grado es mayor de tres por la gran cantidad de sumas, multiplicaciones y divisiones necesarias. Además, si el sistema está mal condicionado, pueden aparecer grandes errores en la solución debido a los problemas de redondeo. No obstante, existen otros métodos que proporcionan una solución con un menor número de operaciones y mayor exactitud (cuando el sistema está mal condicionado), estos métodos están basados en las propiedades de las matrices y los determinantes.

Con este problema tipo se trabajará con estructuras de datos tipo matriz, realizando permutaciones y cambios de columnas, así como se abordará el cálculo de determinantes.

Resolución del problema

A continuación se describe brevemente la regla de Cramer para posteriormente realizar el organigrama del algoritmo que, aplicando dicha regla, resuelve un sistema cualquiera de ecuaciones lineales de orden tres.

Si consideramos el sistema de ecuaciones anterior en forma matricial: $A \cdot x = B$, donde A es la matriz de coeficientes, B la matriz de términos no homogéneos y x la matriz columna de las incógnitas, entonces:

El sistema de ecuaciones tiene solución única si y solo si A es invertible, y en ese caso la solución viene dada por las fórmulas de Cramer:

$$x_i = \frac{\det(A_i)}{\det(A)}, \text{ para i} = 1,2,...,\text{n}$$

donde *x* son las incógnitas del sistema (i-ésima componente de la matriz columna *x*), y la matriz A_i se obtiene al reemplazar la i-ésima columna de la matriz *A* por la matriz columna *B*.

Resolver un sistema de ecuaciones de orden n requiere calcular el valor de *n+1* determinantes de orden *n*. Una correcta y optima codificación de los algoritmos para la evaluación de determinantes de matrices de orden mayor que tres conlleva el uso de algoritmos recursivos complejos, que no son el objeto de este libro. Esta es la razón por la que aquí nos limitaremos a resolver sistemas de ecuaciones de orden tres, que solamente requieren la evaluación de determinantes de tercer orden, cuya fórmula es bien conocida[13]. Para optimizar el algoritmo, se desarrollará un subalgoritmo que calcule el determinante de una matriz de orden tres, en forma de función, llamada DET.

FUNCION DET (a)

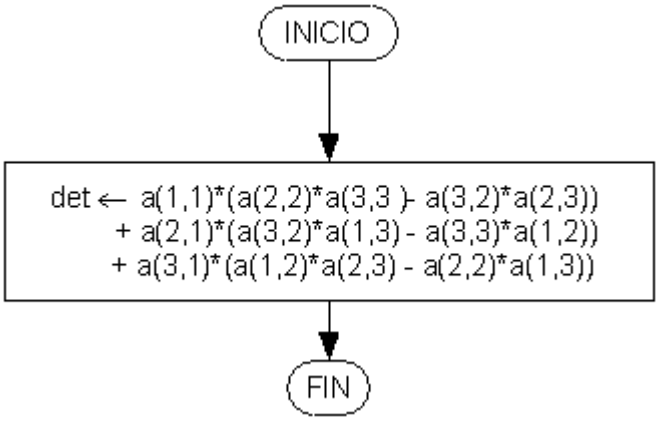
ENTRADA a(3,3): Matriz 3x3 de reales

SALIDA det: real

INICIO

det←a(1,1)*(a(2,2)*a(3,3)-a(3,2)*a(2,3))+a(2,1)*(a(3,2)*a(1,3)-a(3,3)*a(1,2))+a(3,1)*(a(1,2)*a(2,3)-a(2,2)*a(1,3))

FIN



[13] Si $A=\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$, $\det(A)= a_{11}\cdot a_{22}\cdot a_{33}+a_{21}\cdot a_{32}\cdot a_{13}+a_{12}\cdot a_{23}\cdot a_{31}-a_{31}\cdot a_{22}\cdot a_{13}-a_{11}\cdot a_{32}\cdot a_{23}-a_{12}\cdot a_{21}\cdot a_{33}$

El algoritmo de cálculo se limitará a seguir los siguientes pasos:

- Introducir los coeficientes de cada una de las ecuaciones lineales, almacenándolos en la matriz *A* y en el vector *b*.
- Evaluar el determinante de la matriz *A* (*deta*) usando una llamada a la función DET. Si es igual a cero, el sistema de ecuaciones es indeterminado y no se puede obtener una solución.
- Si el sistema es determinado, entonces, de un modo cíclico, se obtienen, una después de la otra, las matrices A_i, almacenándolas en la matriz *d*, se evalúan sus determinantes (*detd*), usando una llamada a la función DET, se obtienen cada una de las soluciones del sistema de ecuaciones (*x*) y se envían al exterior.

La principal dificultad de este algoritmo radica en el manejo de matrices, para ello la opción más clara es usar bucles anidados del tipo PARA, de modo que se pueda recorrer una matriz de un modo ordenado (fijando el índice de las filas y moviendo el de las columnas o viceversa según lo requiera el problema a resolver).

En este caso se utilizan tres bucles PARA anidados, el primero de ellos basado en el contador *i* se utiliza para seleccionar la columna que se va a reemplazar en la matriz *A* por el vector *B* para obtener la matriz A_i. Fijada la columna a reemplazar (i-ésima), se toma la matriz *A* y se recorre por columnas usando la variable contador *j*, en el caso de que *i* sea igual a *j* se reemplaza toda la columna j-ésima por la matriz *B*, en caso contrario no se reemplaza la columna j-ésima. Para reemplazar la j-ésima columna se utiliza otro bucle, esta vez sobre la variable contador *k*, que permite acceder a todos los elementos de la mencionada columna de la matriz *A* y a todos los elementos del vector o matriz columna *B*.

El diagrama de flujo del algoritmo propuesto es:

ALGORITMO Regla de Cramer para la solución de un sistema de ecuaciones lineales de orden 3

ENTRADA Matrices A, B

SALIDA Solución de la ecuación $A \cdot x = B$

VARIABLES

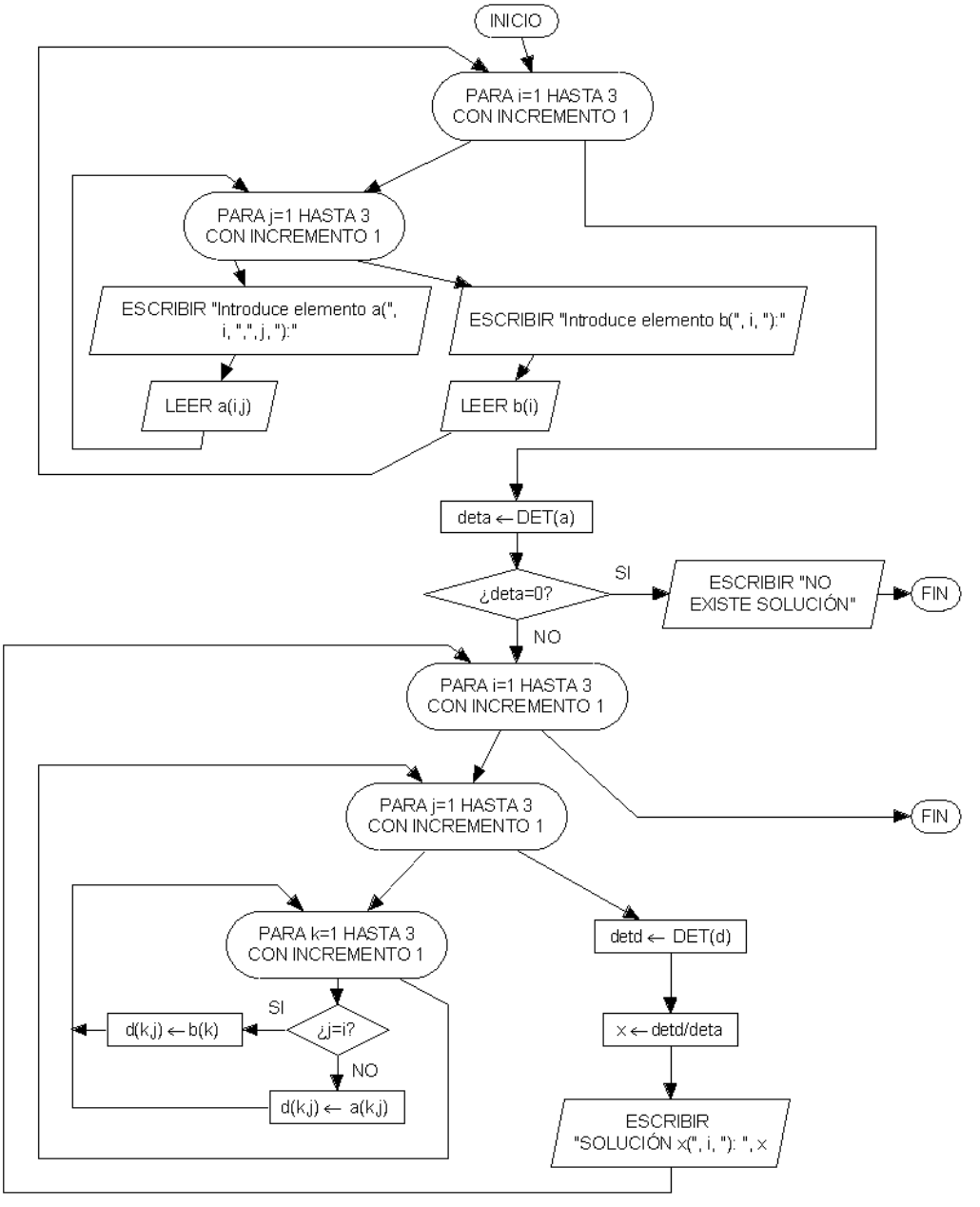
a(3,3), d(3,3), b(3,1): matrices de elementos reales

deta, detd, x: reales

i, j, k: enteros

INICIO

```
        PARA i=1 HASTA 3 CON INCREMENTO 1
                PARA j=1 HASTA 3 CON INCREMENTO 1
                        ESCRIBIR “Introduce elemento a(“, i, “,”, j, “): “
                        LEER a(i,j)
                FIN_PARA
                ESCRIBIR “Introduce elemento b(“, i, “): “
                LEER b(i)
        FIN_PARA
        deta←DET(a)

        SI deta=0.
                ESCRIBIR 'No existe solución'
        SI_NO
                PARA i=1 HASTA 3 CON INCREMENTO 1
                        PARA j=1 HASTA 3 CON INCREMENTO 1
                                PARA k=1 HASTA 3 CON INCREMENTO 1
                                        SI j=i
                                                d(k,j)←b(k)
                                        SI_NO
                                                d(k,j)←a(k,j)
                                        FIN_SI
                                FIN_PARA
                        FIN_PARA
                        detd←DET(d)
                        x←detd/deta
                        ESCRIBIR “Solución x(“, i, “): ”, x
                FIN_PARA
        FIN_SI
FIN
```



Como prueba del algoritmo resuelva el siguiente sistema de ecuaciones:

$$\begin{pmatrix} 1 & 2 & 0 \\ -1 & 1 & 1 \\ 1 & 2 & 3 \end{pmatrix}\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}.$$

Nótese que $\begin{vmatrix} 1 & 2 & 0 \\ -1 & 1 & 1 \\ 1 & 2 & 3 \end{vmatrix} = 1\cdot(1\cdot 3 - 1\cdot 2) - (-1)\cdot(2\cdot 3 - 0\cdot 2) + 1\cdot(2\cdot 1 - 1\cdot 0) = 9$, lo que implica que la matriz A es invertible y por tanto se puede aplicar la regla de Cramer

resultando que: $x=\frac{1}{9}\begin{vmatrix}0 & 2 & 0\\ 1 & 1 & 1\\ 0 & 2 & 3\end{vmatrix}=\frac{-6}{9}=\frac{-2}{3}$, $y=\frac{1}{9}\begin{vmatrix}1 & 0 & 0\\ -1 & 1 & 1\\ 1 & 0 & 3\end{vmatrix}=\frac{3}{9}=\frac{1}{3}$,

$z=\frac{1}{9}\begin{vmatrix}1 & 2 & 0\\ -1 & 1 & 1\\ 1 & 2 & 0\end{vmatrix}=0.$

6.2.5 Resolución de una integral definida

Existen muchos problemas científicos y técnicos en los que es necesario evaluar la integral definida de una función ($\int_a^b f(x)\cdot dx$). Algunos ejemplos son:

- El cálculo del trabajo desarrollado por una fuerza cuando se desplaza por una trayectoria desde un punto $\bar{r}_1$ a otro $\bar{r}_2$: $W=\int_{\bar{r}_2}^{\bar{r}_1}\bar{F}(\bar{r})\cdot d\bar{r}$.

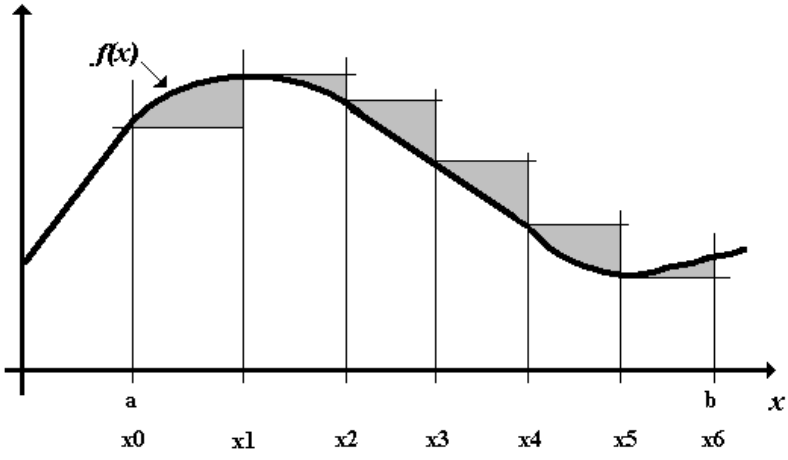
- El cálculo del valor medio de una magnitud que evoluciona a lo largo del tiempo según una ley $y(t)$, siendo $\bar{y}=\frac{1}{t_2-t_1}\int_{t_1}^{t_2} y(t)\cdot dt$.

- Un campo en el que es imprescindible el cálculo de las integrales definidas es en el de la simulación por ordenador de sistemas continuos, en el que dado un conjunto de ecuaciones algebraico-diferenciales que representan la dinámica de un sistema cualquiera (modelo matemático), estas pueden resolverse de modo que dado el estado del sistema en tiempo t=0 puede conocerse el estado del sistema en cualquier instante posterior.

  En general el modelo dinámico de un sistema continuo es de la forma: $\begin{cases}F(\dot{x},x,y,t)=0.\\ G(x,y)=0.\end{cases}$, donde $\dot{x}=\frac{dx}{dt}$. Como no siempre se puede encontrar una solución analítica de la forma $\begin{matrix}x=x(t)\\ y=y(t)\end{matrix}$ debe recurrirse a métodos numéricos que permitan obtener los valores de *x* e *y* en sucesivos y equidistantes instantes de tiempo desde el instante inicial hasta el final. Estos métodos numéricos están basados en la evaluación de integrales definidas.

La integral definida $\int_a^b f(x)\cdot dx$, donde $f(x)$ es una función continua en el intervalo (*a*, *b*), puede ser interpretada geométricamente como el área bajo de la curva $y=f(x)$ entre $x=a$ y $x=b$. Si dicho intervalo se subdivide en *n* subintervalos (x_i, x_{i+1}) de longitud *h*, entonces esa integral definida puede

aproximarse por la suma de las áreas de los *n* rectángulos de altura $f(x_i)$ y anchura *h* que resultan al dividir el intervalo (*a*, *b*) de la forma descrita. Dicha suma puede calcularse aplicando la fórmula $\sum_{i=0}^{i=n-1} f(x_i)\cdot h$, que coincidirá con el valor de la integral definida cuando $n\rightarrow\infty$.



Evidentemente existen otros muchos métodos numéricos que permiten evaluar la integral definida de una función, existiendo algoritmos que estiman el error que se está cometiendo al hacer la aproximación y son capaces de elegir una u otra fórmula de cálculo, así como variar el tamaño del paso de integración en función del error cometido. Por supuesto, la precisión aportada por estos métodos se contrapone a la complejidad de cálculo asociada. En este libro no se pretende hacer una descripción de dichos métodos, tan sólo mostrar cómo se puede resolver una integral definida usando un ordenador.

Para ilustrar el cálculo de integrales definidas, a continuación se plantea el cálculo de la temperatura media de una habitación entre dos instantes de tiempo dados cuya temperatura evoluciona a lo largo del tiempo según la ley $T(t) = 50.\left(e^{-t} + 0.1\cdot sin(10.*t)\right)$, siendo *T* la temperatura en ºC y *t* el tiempo en horas.

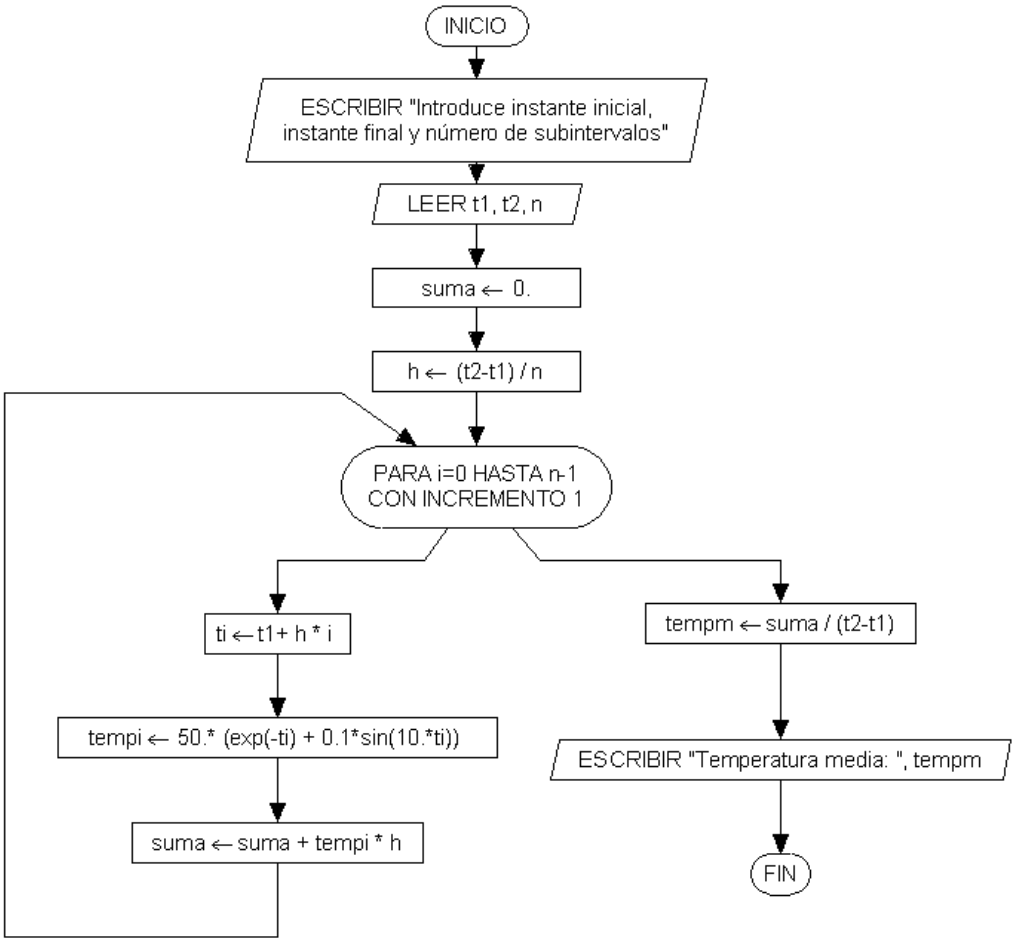
Resolución del problema

La solución del problema resulta ser $\overline{T} = \frac{1}{t_2 - t_1}\int_{t_1}^{t_2} 50.\left(e^{-t} + 0.1\cdot sin(10.*t)\right)dt$. El cálculo de la magnitud $\overline{T}$ requiere determinar el valor de una integral definida, para ello se desarrollará un algoritmo basado en el cálculo de una integral usando la aproximación por rectángulos particularizado a la función que describe la evolución de la temperatura.

El algoritmo solicitará los tiempos inicial (*t1*) y final (*t2*) del intervalo en el que se desea calcular la temperatura media y el número de subintervalos (*n*) en los que se va a dividir dicho intervalo para aproximar el cálculo de la integral. El paso de integración se denominará *h*, y el valor de la integral se almacena en la variable *suma*, que debe inicializarse a cero en el inicio del algoritmo. Para el cálculo de la integral se requiere un bucle tipo PARA con una variable tipo contador (*i*) y otra tipo acumulador (*suma*). En el bucle PARA primero se generan los tiempos que aparecen al subdividir el intervalo temporal en el que se desea calcular el valor medio de la temperatura (*ti*), el valor de la temperatura en cada uno de esos instantes (*tempi*) y se acumula el área del rectángulo, cuya base es *ti* y su altura *tempi*, en la variable *acumulador*. Al salir del bucle PARA se estimará el valor medio de la temperatura (*tempm*) y se presentarán resultados.

Nótese que para estimar el valor de *tempi* se usan dos funciones externas *exp* y *sin* que están definidas en la gran mayoría de los lenguajes de programación.

```
ALGORITMO Cálculo de la temperatura media de una habitación
ENTRADA Intervalo temporal, número de pasos del algoritmo integración
SALIDA La temperatura media
VARIABLES
        t1, t2, h, suma, ti, tempi, tempm: reales
        i, n: enteros
INICIO
        ESCRIBIR "Introduce el instante inicial, instante final y número de
        subintervalos"
        LEER t1,t2,n
        suma ← 0.
        h ← (t2-t1)/n
        PARA i=0 HASTA n-1 CON INCREMENTO 1
                ti ← t1+h*i
                tempi ← 50.*(exp(-ti)+0.1*sin(10.*ti))
                suma ← suma+tempi*h
        FIN_PARA
        tempm ← suma/(t2-t1)
        ESCRIBIR "Temperatura media: ", tempm
FIN
```


INICIO
ESCRIBIR "Introduce instante inicial, instante final y número de subintervalos"
LEER t1, t2, n
suma ← 0.
h ← (t2-t1) / n
PARA i=0 HASTA n-1 CON INCREMENTO 1
ti ← t1+ h * i
tempi ← 50.* (exp(-ti) + 0.1*sin(10.*ti))
suma ← suma + tempi * h
tempm ← suma / (t2-t1)
ESCRIBIR "Temperatura media: ", tempm
FIN

7. Lenguaje C

El lenguaje C viene en los últimos años asumiéndose como el de propósito general más indicado para la programación científica. Contribuyen a ello su capacidad de acceso a bajo nivel (sistemas operativos y drivers de dispositivos están escritos en este lenguaje) incluso para operaciones de lógica con registros, y su generación dinámica de almacenamiento, muy interesante para aplicaciones que despliegan su funcionalidad bajo configuración. Esta última propiedad está estrechamente relacionada con el concepto de puntero y la capacidad de declarar arrays sin tamaño predefinido. Además, los operadores y funciones matemáticas admiten tratamientos sencillos, con lo que la implementación de algoritmos alcanza una gran potencia.

7.1 Tipos de datos en lenguaje C

Las dos operaciones básicas relacionadas con datos son:

> Declaración: "Presentación de una variable que se va a utilizar en un programa, estableciendo el tipo al que pertenece"

que lleva asociada una reserva de memoria acorde con su tamaño

> Inicialización: "Operación de asignación de un valor inicial a las variables"

La inicialización es importante porque una variable sin inicializar puede tener valores por defecto (normalmente cero o valores negativos muy grandes) que pueden dar lugar a comportamientos erráticos e imprevisibles.

Ambas operaciones pueden aparecer juntas en una misma sentencia, aunque una buena práctica de programación estructurada aconseja separarlas en localizaciones dedicadas:

- Declaración en ficheros de cabecera (header), con extensión .h, que son accedidos desde el fichero principal mediante una sentencia de preprocesador #include[14].
- Inicialización en la región inicial de los ficheros de código con extensión .c

[14] El preprocesador es un elemento que actúa en los estadíos iniciales de la compilación para desplegar el contenido de ficheros o sustituir el valor de constantes en los puntos en que se los invoca

Las constantes simbólicas se pueden considerar un caso particular de variables con valor de inicialización no modificado posteriormente a lo largo del programa.

C es un lenguaje altamente prototipado (*typed*), en el sentido de que el programador debe proveer de forma explícita un tipo de dato (*type*) para cada nombre de variable utilizada en un programa. Esto le diferencia de otros lenguajes que como Matlab pueden hacer declaraciones implícitas en el acto de inicialización (equivalente a una imprimación). El estándar especifica, además, que las declaraciones deben situarse de forma agrupada en cada función antes del código propiamente dicho.

7.1.1 Nombres de variables

Deben observarse unas mínimas reglas de denominación para facilitar la comprensión del significado de las variables utilizadas. Así, pueden ser alfanuméricas (se pueden componer de letras y dígitos), pero en todo caso el primer carácter debe ser una letra. No pueden dejarse espacios intermedios, y para nombres compuestos de varias palabras, por motivaciones semánticas, se puede hacer uso de la barra baja '_' para componerlas. Además, se distingue entre letras mayúsculas y minúsculas (*case sensitive*)

Existen reglas más restrictivas derivadas de convenios adoptados por grupos más o menos grandes de programadores para la denominación de variables y con el fin de dotar de una componente semántica al propio nombre (ej.: notación húngara), generalmente basadas en clasificaciones. En general, el criterio sería que dos programadores distintos del mismo grupo, ante la necesidad de declarar una determinada variable en un contexto clasificable, le darían el mismo nombre. En general, unas reglas de buen uso de denominaciones pasarían por:

- Asignar a las variables nombres que guarden relación con su significado o justificación (p.ej, SalAnalog, EntDig)
- Utilizar minúsculas para nombres de variables y mayúsculas para nombres de constantes simbólicas
- Evitar nombres cortos genéricos
- Evitar palabras clave (reservadas) y nombres demasiado similares que puedan dar lugar a errores tipográficos

7.1.2 Tipos y tamaños de datos

Aunque los compiladores pueden mejorar el número de bytes asignados a cada tipo de dato, y siempre que se respeten los tamaños relativos entre ellos, los tamaños mínimos que deben asegurarse son:

char: Un byte que contiene un carácter del conjunto alfanumérico

int: Generalmente en dos bytes, entero comprendido entre [-32768, 32767] (resultado de $\left[-2^{n-1}, 2^{n-1}\right], n = 16$).

float: Generalmente en cuatro bytes, real en punto flotante con precisión normal

double: Generalmente en ocho bytes, real en punto flotante con doble precisión

Calificadores a int, que de hecho lo sustituyen:

- *short*: Valor por defecto, generalmente dos bytes
- *long*: Generalmente cuatro bytes

Calificadores a char o a cualquier entero

- *signed*: Positivos y negativos
- *unsigned*: Siempre positivos o cero

Por ejemplo: signed char en el intervalo [-128, 127], unsigned char en el intervalo [0, 255].

ANSI C no tiene un tipo de dato lógico, pero los compiladores lo suelen incorporar, y en cualquier caso, es inmediato añadirlo en preprocesador, con tamaño de un bit y valores posibles 0/1 ó True/False.

Ejemplos de declaración de variables son:

```
double resuloperan;
int contador = 1;  /* con inicialización */
```

7.1.3 Declaración de constantes

Se puede hacer exactamente igual que las variables (se convierten en constantes por el hecho de no reasignar valores a lo largo del programa). También se puede utilizar el cualificador *const* en la declaración para indicar que no se puede cambiar de valor, de modo que si aparece posteriormente en asignaciones dará lugar a un error de compilación. Ejemplos de declaración de constantes son:

```
const double pi;
const int gr[] = {17, 23, 54, 73};
```

Para marcar su carácter de constante, se puede utilizar la sentencia #define del preprocesador, que en la práctica hace una sustitución en tiempo de compilación

```
#define TAMAÑOMAX 50
```

Las constantes de tipo carácter van entre apóstrofos. Ej.: `char 'a'`.

Si se trata de una cadena de caracteres, debe ir entre comillas. Ej.: `char close[5]="Salir"`.

Ciertos caracteres no representables gráficamente (caracteres de control) son considerados mediante secuencias de escape. Ej.: `'\n'` para nueva línea.

Las constantes se pueden agrupar en forma de enumeraciones de la forma:

```
enum dia {lunes, martes, miercoles, jueves, viernes,
sabado, domingo};
```

(de modo que se crea un nuevo tipo de dato en que sólo se puede tomar alguno de los valores señalados)

7.1.4 Conversiones de tipo implícitas y explícitas (casting)

Una conversión de tipo implícita se produce

- Cuando se opera con variables de distinto tipo: ante la necesidad de que sean del mismo tipo para poder operar con ellas, se promociona la de menor tamaño al tipo de la de mayor tamaño.
- Cuando tiene lugar una asignación y el tipo de la variable a la izquierda de la igualdad impone un tamaño (ampliando o truncando) al resultado contenido en la expresión a la derecha.

El casting o conversión de tipo explícita incluye el tipo de dato a que se desea forzar un resultado acompañando a este:

```
float nivel;
flotacion = (int) nivel + (int) 3.4
```

Se utiliza sobre todo para acomodar valores de retorno de funciones que son tratados en distintos casos a los que deben ajustarse.

7.1.5 Operadores

Los operadores establecen los mecanismos de tratamiento y manipulación de las variables, su relación entre ellas y con constantes, y las modificaciones afectándolas por separado.

Los operadores aritméticos son los más utilizados en algoritmos, y se distingue entre:

binarios: +, -, *, /, operador módulo %.

unarios: + (signo), - (signo).

El operador módulo se utiliza para tomar decisiones en función del resto de un cociente. En la expresión x%y produce el resto del cociente entre las variables x, y (es cero cuando x es múltiplo de y). No puede aplicarse a operandos de tipo float o double. En el ejemplo:

```
if ((year % 4 == 0 && year % 100 != 0) || year % 400 ==0)
 printf ("%d es un año bisiesto\n", year);
else
 printf("%d no es un año bisiesto\n", year);
```

se distinguen los años bisiestos y sus excepciones de una forma inmediata utilizando el operador módulo (recordar el ejemplo de la página 36 en capítulo 2).

La llamada **precedencia de operadores** establece el orden de ejecución de operaciones en expresiones compuestas, es decir, qué operadores se aplican primero para obtener un operando resultado al que aplicar en nuevas operaciones. El orden de precedencia es:

1. +, - unarios
2. *, /
3. +, - binarios
4. %

Los paréntesis () pueden utilizarse para establecer precedencias alternativas o remarcar las precedencias por defecto.

Ej.:

```
a=1, b=2, c=3;
d = 3*a+7;          /* d = 10 */
e = 3*(a+7);        /* e = 24 */
f = 4+c-b*2;        /* f = 3 */
```

Por su parte, los operadores de relación se utilizan en expresiones basadas en inecuaciones o evaluaciones. Entre ellos se encuentran:

1. mayor >
2. mayor o igual >=
3. menor <
4. menor o igual <=
5. igual ==
6. distinto !=

Todos ellos con menor precedencia que los aritméticos:

`i < lim-1` equivale a: `i < (lim-1)`

Los operadores lógicos también pueden ser unarios (NOT se expresa como !) y binarios (AND que se expresa como &&, OR que se expresa como ||). Su precedencia de mayor a menor es: NOT, AND, OR.

Los **operadores de incremento y decremento** son operadores unarios que pueden ser utilizados como prefijos y como sufijos. En el siguiente segmento de código aparecen junto a su significado entre comentarios:

```
int n1=5, x;
 x=++n1        /* incrementa n1 antes de utilizar su
               valor, x=6 */
 x=--n1        /* decrementa n1 antes de utilizar su
               valor, x=4 */
 x=n1++      /* incrementa n1 después de utilizar su
             valor,  x=5 */
 x=n1--      /* decrementa n1 después de utilizar su valor
             x=5 */
```

También pueden aparecer en expresiones directas sin asignación:

```
++n1;
```

Los **operadores de asignación** pueden compactarse en determinados casos. Por ejemplo:

`i=i+2` es equivalente a: `i+=2`

En general, los operadores binarios tienen un operador de asignación correspondiente de la forma *op*=, donde *op* puede ser +, -, *, /, %. Así, las dos asignaciones siguientes son equivalentes:

```
var op= expr
var = var op (expr)
```

7.1.6 Expresiones:

> Las expresiones son combinación de variables, constantes, valores constantes, funciones y operadores para obtener nuevos valores o resultados

El caso más común de expresión con resultado aritmético es un algoritmo de cálculo, mientras que el caso más común de expresión con resultado lógico es la evaluación de la ecuación o inecuación entre paréntesis en un control de flujo

```
Si X = 1 (verdadero), Y = 0 (falso), A = 1, B = 2, C = 3
A<B es verdadero
B>C es verdadero
```

```
! X && (A>=C) || (B!=C) || ! (X || Y) es verdadero
```

7.1.7 Tipos de dato compuestos

> Array: "Estructura de datos formada por una cantidad fija de datos de un mismo tipo, cada uno de los cuales tiene asociado uno o más índices que determinan de forma unívoca la posición del dato"

Se declara aportando tamaño entre corchetes a un tipo de dato básico, como int z[10]; o float comp[2] = {1.4, 3.7}. El tamaño de un array es el número total *n* de elementos que contiene, aunque para referirse a ellos de forma particularizada se cuenta desde *0* hasta *n-1*. Así, en los arrays dados estarían contenidos los elementos enteros z[0], z[1], ..., z[9] y los reales de precisión normal comp[0]=1,4, comp[1]=3.7, respectivamente.

Existe una relación muy estrecha entre los elementos de un array y la forma de acceso mediante punteros. Los punteros son una herramienta de una gran potencia de la que carecen la mayor parte de lenguajes de programación. Un puntero permite acceder a la dirección en que se almacena una variable, y será del tipo de dato de dicha variable.

Operadores unarios complementarios:

*: puntero o dirección (acompaña a un puntero para asignar un valor a la variable apuntada)

&: la dirección de, acompaña a una variable para asignar un puntero

Ejemplo:

```
int x = 1, y = 2, z[10];
int *ip;     /*  ip es un puntero a int */
ip = &x      /*  ip apunta a x */
y = *ip;     /*  y vale 1 */
*ip = 0;     /*  x vale 0 */
ip = &z[5];  /*  ip apunta al sexto elemento de z[10] */
```

Cadena de caracteres, string o array:

Son un caso particular de array formado por una secuencia de caracteres en un orden determinado, como: `char frase[100];`

Existe toda una serie de funciones de comparación, concatenación, búsqueda y tratamiento de cadenas de caracteres, así como de conversión entre resultados numéricos y strings para su adquisición desde dispositivos de entrada o su representación en dispositivos de salida.

Los arrays o cadenas de caracteres son muy importantes porque suponen la implementación de los tratamientos textuales y la generación de respuestas en lenguaje natural en situaciones de interactividad con el usuario. Los caracteres especiales de control y delimitadores de cadena en arrays de caracteres son:

'\n': carácter nueva línea

'\r': carácter retorno de carro

'\0': fin de cadena

'\t': carácter de tabulación

'\\': carácter diagonal invertida (backslash)

EOF: Fin de fichero

Los tratamientos en arrays de caracteres están modularizados en funciones que reciben las cadenas a tratar mediante argumentos que son punteros a ellas. Así por ejemplo, para copiar la cadena *t* en la cadena *s*, la versión básica sería:

```
void strcpy(char *s, char *t)
{int i=0;
 while ( (s[i] = t[i]) != '\0' )
       i++;                    }
```

Una versión avanzada haciendo uso de punteros quedaría:

```
void strcpy(char *s, char *t)
{while ((*s++ = *t++) != '\0')
 ;                       }
```

Otro ejemplo sería la comparación de las cadenas *s* y *t* y sus longitudes con una función que devuelve 0 si *s* y *t* son de la misma longitud, <0 si *s*<*t*, y >0 si *s*>*t*, restando los caracteres de la primera posición en que *s* y *t* no coinciden. La versión básica sería:

```
int strcmp(char *s, char *t)
{int i;
 for (i=0; s[i] == t[i]; i++)
       if (s[i] == '\0')
             return 0;
 return s[i] - t[i];               }
```

Mientras que la versión con punteros quedaría:

```
int strcmp(char *s, char *t)
{for( ; *s == *t; s++, t++)
       if (*s == '\0')
```

```
            return 0;
 return *s - *t;                    }
```

Otras funciones relacionadas con estos tratamientos se dedican a procesamientos textuales. Un programa para contar las ocurrencias de cada dígito, espacios en blanco y el resto de caracteres se puede implementar fácilmente según la codificación siguiente:

```
#include <stdio.h>
main()
{int c, i, nblanco, notro;
 int ndigito[10];
 nblanco=notro=0;
 for (i=0; i<10; ++i)
       ndigito[i] = 0;
 while ( (c = getchar()) != EOF )
       if (c >= '0' && c <= '9')
             ++ndigito[c]
       else if ( c==' ' || c=='\n' || c=='\t' )
             ++nblanco;
       else
             ++notro;
 printf ("dígitos = ");
 for (i=0; i<10; ++i)
       printf("%d", ndigito[i]);
 printf (", espacios blancos = %d, otros = %d\n", nblanco,
notro);}
```

Arrays multidimensionales y matrices:

Los arrays de dimensiones superiores admiten tratamientos acumulativos, de modo que los arrays de dos dimensiones pueden ser tratados como si fueran un array de una dimensión, cada uno de cuyos elementos es un array. Las dimensiones se escriben entre corchetes para cada una de ellas (es decir, no todas compartiendo los mismos corchetes):

```
dias_mes[filas][columnas]
```

En el ejemplo que se muestra más abajo, aparecen dos funciones:

- dia_del_año: Convierte mes y día en el día del año (contando desde el 1 de enero)
- mes_dia: Convierte el día del año en mes y día (a la inversa que el anterior)

La inicialización de variables se establece entre llaves generales y otras para cada dimensión:

```
static char dias_mes [2][12] = {
 {     31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 },
 {     31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 },
 }
int dia_del_año(int año, int mes, int dia)
{int i, bisiesto;
 bisiesto = año%4 == 0 && año%100 != 0 || año%400 == 0;
 for (i = 0; i < mes; i++)
       dia += dias_mes[bisiesto][i];
 return dia;
}
void mes_dia(int año, int diadelaño, int *pmes, int *pdia)
{
 int i, bisiesto;
 bisiesto = año%4 == 0 && año%100 != 0 || año%400 == 0;
 for (i = 0; diadelaño > dias_mes[bisiesto][i]; i++)
       diadelaño-- = dias_mes[bisiesto][i];
 pmes = i;              /* mes */
 pdia = diadelaño;      /* día del mes */
}
```

En este ejemplo, el tipo de dato char `dias_mes` se utiliza para declarar enteros pequeños que no son caracteres.

Estructuras

> Son agrupaciones de variables definibles, inicializables o instanciables de forma conjunta

Así, para especificar las coordenadas de un punto en el plano se declara:

```
struct punto {          int x;
                        int y;      };
```

Una vez declarada, se comporta como un nuevo tipo de dato:

```
struct punto vertice;
struct punto minimo = { 3, 9 };
```

Se hace referencia a cada elemento de una estructura mediante el operador punto '.'

```
vertice.x = minimo.y;
```

7.2 Control de flujo

> Se conoce como control de flujo la especificación optimizada de alternativas, iteraciones y elecciones interactivas en el procesamiento.

En general, en todos los procedimientos se hace uso de llaves {} para modularizar segmentos de código, reforzado por un indentado (conocido también como sangrado) que dé idea gráfica del nivel de anidamiento. También es conveniente en todos ellos dejar especificada una alternativa por defecto (default), que cubra los casos imprevistos.

7.2.1 Estructura selectiva if-else

Se utiliza en caso de ejecuciones alternativas. Un ejemplo típico es el cálculo del máximo:

```
if (num1 > num2)
 max = num1;
else
 max = num2;
```

el módulo else es optativo, apareciendo desde 0 veces a múltiples casos con else if (expresión).

Además, se puede utilizar la expresión abreviada:

`if(expresion)` en vez de: `if(expresion != 0)`

Una alternativa equivalente es la expresión condicional dada por el operador ternario '?:', que tiene la forma:

```
expr1 ? expr2 : expr3
```

En este caso, la expresión expr1 se evalúa en primer lugar. Si es cierta, se evalúa la expresión expr2 para calcular el valor de retorno. Si es falsa, se evalúa la expresión expr3 para calcular el valor de retorno. Para el cálculo del máximo, la utilización de este operador quedaría en la forma:

```
max = (num1 > num2) ? num1 : num2;
```

Anidamiento

Consiste en la aplicación sucesiva de sentencias de este tipo. En el ejemplo del máximo, se puede incorporar una mejora para considerar números positivos:

```
if ( (num1 >0) && (num2 >0) )
 if (num1 > num2)
 max = num1;
 else
 max = num2;
```

por defecto, `else` acompaña a la estructura inmediata más interna. Si se desea lo contrario hay que explicitarlo con ayuda de las llaves. También es aconsejable ayudarse del indentado progresivo para reconocer a simple vista el nivel de anidamiento en cada línea.

7.2.2 Estructura selectiva switch

Está indicado en casos de decisión múltiple con búsqueda de coincidencias en el resultado de una expresión:

```
switch (expresión) {
 case exp-const: proposiciones
 case exp-const: proposiciones
...
default: proposiciones   }
```

Se produce un translado del control al case en que se produce la coincidencia. Las proposiciones en el segmento default se ejecutan en ausencia de coincidencias. El número de segmentos case y la presencia de segmento default son optativos.

Después de ejecutar un segmento case se continúa en el siguiente, aunque lo normal es plantear las alternativas como excluyentes con ayuda de la sentencia *break*, que provoca una salida inmediata del switch.

7.2.3 Bucle while, do-while

Consiste en una comprobación en cada iteración para decidir si se continúa, sin número preestablecido de iteraciones

En su forma pre-test se fuerza una comprobación previa:

```
while (expresión)
 proposición
```

Ej.:

```
#define DIA 86400
periodo=...
while (tiempo_total < DIA)
{numero_orbita++;
```

```
 tiempo_total+=periodo;  }
```

En su forma post-test, la comprobación es posterior (al menos se ejecuta una vez):

```
do
 proposición
while (expresión)
```

Ej.:

```
...
do
{numero_orbita++;
 tiempo_total+=periodo;  }
while (tiempo_total < DIA)
```

7.2.4 Bucle for

Consiste en una iteración con número preestablecido dado por un índice incrementable o decrementable. Las condiciones de iteración vienen dadas por tres expresiones:

```
for (expr1; expr2; expr3)
 proposición
```

Relación con bucles while: Serían equivalentes para el caso:

```
expr1;
while (expr2)      {     proposición
             expr3;            }
```

expr1 y expr3 son asignaciones, eventualmente incluyendo invocaciones a funciones.

expr2 es una expresión de relación. En un caso típico:

```
for (cont=0; cont<max; cont++)
```

Ejemplo: Inversión de una cadena de caracteres (“libro” se transformaría en “orbil”)

```
#include <string.h>
void reverse(char s[])
{
 int car, contadelante, contatras;
 for(contadelante=0, contatras=strlen(s)-1;
 contadelante<contatras; contadelante++,contatras--)
 {     car = s[contadelante];
```

```
            s[contadelante] = s[contatras];
            s[contatras] = car;                }
      }
```

Cualquiera de las tres expresiones puede omitirse a favor de su alternativa por defecto, conservando el ';' para respetar el orden entre expresiones. El bucle for sin expr2 se supone cierto siempre. Por su parte, for(;;) es una iteración infinita, que generalmente se utilizará como base de una iteración básica con salida por otros medios (break o return)

7.3 Entrada y salida formateada

Contiene dos tipos de objetos: Caracteres ordinarios que son copiados al flujo de salida o del flujo de entrada, y especificaciones de conversión que empiezan por la cadena de conversión '%letra'. Hay una especificación de conversión para cada argumento posterior y es dependiente de su tipo de dato. Entre las cadenas de conversión se encuentran:

%o	octal
%x	hexadecimal
%c	carácter
%s	cadena de caracteres
%d *y* %i	entero
%f	float
%h	short
%l	long

En muchos casos, entre el % y el carácter de conversión puede haber modificadores, que establecen el ancho mínimo del campo o el número de dígitos después del punto decimal, en caso de números reales. Así, por ejemplo, %10.3f nos diría que se imprimirá el dato real con un ancho de 10 posiciones y con tres dígitos después del punto decimal.

7.3.1 Función de salida printf

Recoge resultados de programa almacenados en variables especificadas en los argumentos y los envía a la salida. El flujo de salida se envía al dispositivo por defecto (normalmente la consola en pantalla) o se redirige a otros dispositivos con funciones análogas (ficheros, ventanas). Así por ejemplo:

```
printf("Hola");    /* escribe Hola en la consola" */
printf("El area es %f\n", area);
printf("El area es %.2f\n", area);
```

```
printf("El area es %6.2f\n", area);
printf("%6i %10.3f\n", altura, presion);
fprintf(outp, "El area es %f\n", area);
```

7.3.2 Función de entrada scanf

Recoge información introducida por el usuario y almacena el resultado en las variables especificadas en los argumentos correspondientes. Los argumentos de variables deben ser punteros a las mismas. El flujo de entrada se recoge en el dispositivo por defecto (normalmente el teclado) o se redirige a otros dispositivos con funciones análogas (ficheros, ventanas). Así por ejemplo;

```
scanf("%f", &radio);
fscanf(inp, "%f", &radio);
```

7.4 Funciones y la estructura del programa

Las funciones permiten modularizar tareas grandes en varias pequeñas y son la base de la posibilidad de reutilización. Además, ocultan detalles de operación y su utilización de forma transparente, clarifican las grandes líneas de ejecución y facilitan la introducción de modificaciones. La modularidad se ve reforzada mediante la localización en ficheros propios que admiten precompilación en el caso de las funciones de biblioteca. En general, en una función se distinguen:

- Argumentos: Variables auxiliares que acompañan entre paréntesis al nombre de la función y se necesitan para modificar y especificar su funcionamiento. Se puede hacer la declaración de argumentos y valores de retorno juntamente con los de la función

```
tipofun funcion (tipo arg1, tipo arg2)
{tipofun var;
 ...
 return var;       }
```

- Valores de retorno: Se devuelven una vez elaborados en los distintos puntos del programa (puede haber más de uno) con la palabra clave *return*.

```
double valor_abs(double x)
{if (x<0)
       return -x;
 else
       return x;   }
```

Como caso particular, todo programa en lenguaje C debe contar con una función main, que es la que se ejecuta en primer lugar y desde la que se invoca al resto de funciones.

Los argumentos de la función main están relacionados con los modificadores con que se invoca a la aplicación (nombre de la función en vez de main) desde línea de comandos. Por su parte, el valor de retorno es declarable al principio de su definición y acorde con el tipo devuelto con su return.

Para poder utilizar una función debe aparecer previamente a la función main en que se utiliza en una de las siguietes alternativas: O como prototipo de la función en forma de declaración explícita de la función; o como función con todo su código situada con antelación.

7.4.1 Argumentos: Llamadas por valor

Si es necesario actualizar un valor de una variable utilizando una función, se puede invocar directamente en una asignación utilizando su valor de retorno. Así por ejemplo:

```
resultado = potencia(2, 3);          /* resultado = 8 */
```

Siendo:

```
int potencia(int base, int n)
{int p;
 for(p=1; n>0; --n)
       p=p*base;
 return p;         }
```

En el paso de argumentos por valor hay que tener en cuenta que:

- Las funciones reciben los valores de los argumentos, no las variables originales.
- La función puede modificar variables que se le pasan, pero sólo su copia.
- Si es necesario modificar alguna variable que se le pase, hay que proporcionar la dirección de memoria de la variable mediante un puntero a esta.

El paso de argumentos por valor evita que se afecte a los argumentos en la invocación de la función: este hecho, que en general permite preservar el carácter instrumental de las funciones, supone un inconveniente en casos en que se quiere que la llamada a la función afecte directamente a los argumentos. Así por ejemplo, no es suficiente escribir:

```
int i = 3, j = 4;
intercambio(i, j);
```

```
a = i-1; b= j+1;   /* a=2, b=5 (i, j no tienen los valores
                   intercambiados) */
```

para intercambiar los valores asignados a las variables i, j con la función:

```
void intercambio(int i, int j)
{      int interm;
 interm = i;
       i = j;
       j = interm; }
```

Si se requiere una actuación directa sobre los argumentos que aparecen en la invocación es necesario acceder con punteros a la dirección de memoria de las variables utilizadas:

```
int i = 3, j = 4;
intercambio(&i , &j);
a = i-1; b= j+1;  /* a=3, b=4 (i, j tienen los valores
                  intercambiados) */
```

haciendo uso de la función:

```
void intercambio(int *px, int *py)
{      int interm;
       interm = *px;
       *px = *py;
       *py = interm;     }
```

Para utilizar un array como argumento basta con dar la dirección de memoria del principio del array (en este caso, es simplemente el nombre del array), pudiéndose alterar los valores del array.

7.4.2 Alcance

Funciones y variables en un programa no necesitan compartir localización y ser compiladas en el mismo acto. Así, se pueden tener funciones y agrupaciones de código en distintos ficheros fuente (con extensión .c o extensión .h), del mismo modo que se pueden cargar funciones y rutinas compiladas previamente o pertenecientes a bibliotecas. Se conoce como alcance:

> alcance de una variable o nombre de función: "la parte del programa en que es reconocido"

Alcance de variables

Se conoce como variables externas a las declaradas fuera de las funciones y compartidas por todas ellas, de modo que se elimina la declaración directa en los argumentos y se hace innecesaria la utilización de *return* porque el valor que quede al final de la ejecución de la función es observable desde fuera.

Por su parte, las variables estáticas son declaradas con la palabra clave *static* y limitan el acceso a funciones de su mismo archivo.

Las variables registro son declaradas con la palabra clave *register* e indican al compilador que una variable se utilizará muy frecuentemente, de modo que se procuran colocar en registros del hardware y permiten accesos más rápidos. Su comportamiento es distinto en función del tipo de hardware, e incluso pueden ser ignoradas en determinadas arquitecturas.

Una característica muy peculiar de las funciones es que pueden ser invocadas de forma *recursiva* desde sí mismas. Así por ejemplo, para el cálculo del factorial de un número natural se puede utilizar la función:

```
long factorial(int n)
{if (n<=1)   return 1;
 else        return n*factorial(n-1);     }
```

Las variables declaradas dentro de una función se denominan *locales* y tienen el alcance de dicha función, de modo que variables locales con el mismo nombre y pertenecientes a funciones diferentes no colisionan. Los parámetros o argumentos de una función son variables locales.

El alcance de una variable o función externa comprende desde el punto en que se declara hasta el fin del archivo en que reside. Se necesita una declaración *extern* para las variables declaradas en ficheros fuente diferentes a los que la utilizan o si se hace referencia a ellas con anterioridad a su declaración. Así por ejemplo:

(fichero 1)

```
extern int var1;
extern double val[];
void funcion1(int arg) { ... }
double funcion2(void) { ... }
```

(fichero 2)

```
int var1 = 0;
double val[MAXVAL];
```

En este ejemplo, las funciones funcion1() y funcion2() pueden utilizar directamente var1 y val[] en el fichero 1. La declaración con extern expone las

propiedades de una variable en cuanto al tipo, y la inicialización en el fichero 2 supone además una reserva de espacio para su almacenamiento

7.5 Bibliotecas estándar

Se trata de bibliotecas incluidas en el estándar ANSI C y por tanto independientes de compilador, que agrupan funciones auxiliares por tipos de tratamientos. Cuando se está programando y se necesita alguno de estos tratamientos, basta con incorporar el fichero de cabecera correspondiente y enlazar (linkar) con la librería asociada. Entre las más comunes cabe destacar:

Entrada y salida: <stdio.h>

Operaciones sobre archivos

Entrada y salida con formato

Entrada y salida de caracteres y directa

Pruebas de clasificación de caracteres: <ctype.h>. Comprobaciones de tipo. Ej.: isalpha(c), isdigit(c), isspace(c), islower(c), isupper(c), isxdigit(c), etc.

Funciones para cadenas: <string.h>

Funciones que comienzan con str. Ej.: *strcpy(s, ct), *strncpy(s, ct), *strcat(s, ct), *strncat(s, ct), strcmp(cs,ct), *strchr(cs, c), *strrchr(cs, c), strlen(cs)

Funciones que comienzan con mem: Ej.: *memcpy(s, ct, n), *memmove(s, ct, n), *memchr(cs, c, n), memset(s, c, n)

Funciones matemáticas: <math.h>. Ej.: sin(x), cos(x), tan(x), asin(x), acos(x), atan(x), sinh(x), cosh(x), tanh(x), exp(x), log(x), log10(x), fabs(x)

Funciones de utilería: <stdlib.h>. funciones para conversión numérica, asignación de memoria y otras tareas. Ej.: atof(*s), atoi(*s), srand(seed), malloc(size), free(p), exit(status), abs(n), div(num, denom)

Funciones de diagnóstico: <assert.h>. assert(expresión), y assert imprime un mensaje en stderr del tipo:

```
Assertion failed: expresión, file filename, line nnn
```

donde filename y el número de línea vienen dados por __FILE__ y __LINE__ del preprocesador.

Señales: <signal.h>. Manejo de condiciones de excepción aparecidas durante la ejecución del programa, tales como señales de interrupción externas, eventos o errores en la ejecución. Su invocación es:

```
signal(int sig, void(handler)(int))
```

donde sig es la señal a monitorizar (SIGABRT de terminación normal, SIGFPE error aritmético, SIGTERM solicitud de terminación), y handler apunta a una función de tratamiento de la señal.

Funciones de fecha y hora: <time.h>. Tipos (clock_t representa el tiempo, struct tm mantiene las componentes de un calendario) y funciones para manipulación de fecha y hora. Ej.: clock(), time (*tp)

Límites definidos en la implantación: <limits.h>, <float.h>. Constantes mínimas aceptables para el tamaño de los tipos de enteros (pueden ser mayores). Ej.: CHAR_BIT (8), INT_MAX (32767), INT_MIN (-32767)

7.6 Codificación de casos típicos

En este apartado realizaremos la codificación en lenguaje C de los casos típicos presentados en el apartado 6.2. La explicación de la resolución puede encontrarse en dicho apartado, aquí simplemente plantearemos la solución y, en algún caso, haremos alguna consideración adicional.

7.6.1 Sumatorios y medias aritméticas. Centro de masas

```
/*
  ALGORITMO centro de masas
  ENTRADA: número de partículas y datos de cada partícula
  SALIDA:  la masa media y el centro de masas
  VARIABLES:
       k,i: ENTEROS
       MT, MX, MY, MZ: REAL
       C(k,4): matriz de REAL
*/
#include <stdio.h>

main()
{ /* INICIO */
  int k,i;
  float MT, MX, MY, MZ;
  float C[100][4]; /* limitamos el tamaño maximo de C a
                      100 filas */

  /* Leer los datos por teclado */
  printf("Introduce el numero de particulas\n");
```

```
  scanf("%d", &k);

  /* Ahora se deben leer los datos de cada particula */
  for (i=0; i<k; i++)
  {
   printf("Masa de la particula: \n");
   scanf("%f", &C[i][0] );
   printf("Coordenada X: \n");
   scanf("%f", &C[i][1] );
   printf("Coordenada Y: \n");
   scanf("%f", &C[i][2] );
   printf("Coordenada Z: \n");
   scanf("%f", &C[i][3] );
  }
/* Inicializacion de otras variables */
  MT = 0.0;
  MX = 0.0;
  MY = 0.0;
  MZ = 0.0;

  /* Bucle de calculo del centro de masas */
  for (i=0; i<k; i++)
  {
 MT = MT + C[i][0];
 MX = MX + C[i][0]*C[i][1];
 MY = MY + C[i][0]*C[i][2];
 MZ = MZ + C[i][0]*C[i][3];
  }

  /*  Calculos finales */
 MT = MT/k;
 MX = MX/MT;
 MY = MY/MT;
 MZ = MZ/MT;
```

```
 /* Se escribe el centro de masas y la masa media */
printf("\n\nMasa media del sistema de particulas: %f\n",
        MT);
printf("Centro de masas (x,y,z) = (%f,%f,%f)\n\n"
       ,MX,MY,MZ);


} /* FIN */
```

7.6.2 Resolución de ecuaciones no lineales. Método de Newton-Raphson

```
/*            ALGORITMO: método de Newton Raphson
ENTRADAS: cota de error, numero de iteraciones y error
          inicial
SALIDAS: solucion de la ecuacion f(x)=0
*/


#include <stdio.h>
#include <math.h>
#define INFINITO   100000000.0
/* Declaramos las dos funciones a programar */
float funcionf(float x);
float funcionfder(float x);
main()
{/* inicio */
 int i, N;
 float x1, x2, error, cota;

 printf("Numero de iteraciones: \n"); scanf("%d", &N);
 printf("Cota del error: ");          scanf("%f", &cota);
 printf("Solucion inicial: ");  scanf("%f", &x1);

 i=1;
 error = INFINITO;
 while(i<=N && error>cota)
 {
       x2 = x1-funcionf(x1)/funcionfder(x1);
       error = fabs(x2-x1);
```

```
        i++;
        x1 = x2;
 } /* while */
 printf("La solucion a la que se llega es: %f\n", x2);
 printf("El numero de iteraciones es %d \n", i-1);
 printf("El valor final del error es %f \n", error);
 if ((i-1)<N || error<cota)
        printf("Se ha alcanzado una solucion correcta\n");
}/* fin */

float funcionf(float x)   { return x*x+0.5-exp(-x);}
float funcionfder(float x)   { return 2*x+exp(-x);}
```

7.6.3 Operaciones con vectores. Producto escalar y vectorial

```
/*
ALGORITMO: producto escalar y vectorial de dos vectores
ENTRADAS: dos vectores
SALIDAS: el producto escalar, vectorial y el angulo que
         forman ambos
VARIABLES: i: ENTERO
           v1(3), v2(3), productov(3): VECTORES REALES
           productoe, modulov1, modulov2, angulo: REALES
*/

#include <stdio.h>
#include <math.h>

main()
{ /* inicio */
 int i;
 float v1[3], v2[3], productov[3];
 float productoe, modulov1, modulov2, angulo;
 float auxmod;

 /* Inicializacion de variables */
 productoe =  0.0;
```

```
modulov1 =  0.0; /* No haría falta */
modulov2  = 0.0; /* No haría falta */

auxmod = 0.0;

printf("Elementos del vector v1: ");
/* Lectura de los elementos de los vectores v1 y v2 */
/* Lectura de v1 */
for (i=0; i<3; i++)
      scanf("%f", &v1[i]);

printf("Elementos del vector v2: ");
/* Lectura de v2 */
for (i=0; i<3; i++)
      scanf("%f", &v2[i]);

/* Calculamos el modulo de v1 */
for (i=0; i<3; i++)
      auxmod += v1[i]*v1[i];
modulov1 = sqrt(auxmod);

/* Calculamos el modulo de v2 */
/* Hay que reinicializar auxmod */
auxmod = 0.0;
for (i=0; i<3; i++)
      auxmod += v2[i]*v2[i];
modulov2 = sqrt(auxmod);

/*  Producto escalar */
for (i=0; i<3; i++)
  productoe += v1[i]*v2[i];

/* Angulo que forman los dos vectores */
      angulo = acos(productoe/(modulov1*modulov2));
```

```
 /* Producto vectorial */
 productov[0] = v1[1]*v2[2] - v1[2]*v2[1];
 productov[1] = v1[2]*v2[0] - v1[0]*v2[2];
 productov[2] = v1[0]*v2[1] - v1[1]*v2[0];

 /*  Presentacion de resultados al usuario */
 printf("Producto escalar: %f\n", productoe);
 printf("Modulo vector v1: %f\n", modulov1);
 printf("Modulo vector v2: %f\n", modulov2);
 printf("Angulo que forman los dos vectores: %f\n",
angulo*180/M_PI);
 printf("Producto vectorial: %f %f %f \n",
       productov[0], productov[1], productov[2]);
} /* FIN */
```

7.6.4 Resolución de sistemas de ecuaciones lineales. Regla de Cramer

```
/*
 ALGORITMO regla de Cramer
 ENTRADAS: matriz A y vector b
 SALIDA: la solucion del sistema de ecuaciones
 VARIABLES: A(3,3), b(3), d(3,3), deta, detd,x: REALES
            i,j,k: ENTERO
*/

#include <stdio.h>

/* declaracion de la funcion determinante */
float determinante();

main()
{ /* inicio */

 float a[3][3], b[3], d[3][3], x[3];
 float deta, detd;
 int i, j, k;
```

```
 /* Se introducen los datos de la matriz A */
 puts("Introduce los datos de la matriz A");
 for (i=0; i<3; i++)
       for (j=0; j<3; j++) {
         printf("Componente (%d,%d): ",i,j);
         scanf("%f", &a[i][j]);
       }
 /* Introducir b */
 puts("Introduce los datos del vector de terminos
independientes");
 for (i=0; i<3; i++) {
       printf("Componente %d: ",i);
       scanf("%f", &b[i]);
 }

 /*  Se pasa a calcular el determinante de a */
 deta = determinante(a);

 if(deta != 0)
 {
 for (i=0; i<3; i++)
 {
       for (j=0; j<3; j++)
             for (k=0; k<3; k++)
                   if (i==j)
                    d[k][j] = b[k];
                   else
                    d[k][j] = a[k][j];
                   /* end  if */
             /* end for k */
       /* end for j */
      detd = determinante(d);

   x[i] = detd/deta;
 } /* end for i */
```

```
 printf("La solucion del sistema es: %f %f %f \n",
       x[0], x[1], x[2]);
 } /* end if then */
 else
       printf("\nEl determinante de A es cero, asi que el
sistema no tiene solucion\n");
 /* end  if else */
} /* fin */

float determinante(matriz)
float matriz[3][3];
{ /* inicio */
 float aux;

 aux = matriz[0][0]*matriz[1][1]*matriz[2][2]+
         matriz[0][1]*matriz[1][2]*matriz[2][0]+
         matriz[1][0]*matriz[2][1]*matriz[0][2]-
         matriz[2][0]*matriz[1][1]*matriz[0][2]-
         matriz[1][0]*matriz[0][1]*matriz[2][2]-
         matriz[2][1]*matriz[1][2]*matriz[0][0];

 return aux;
} /* fin */
```

7.6.5 Resolución de una integral definida

```
/*
  ALGORITMO: resolución de una integral definida
  ENTRADAS: [t1,t2] y n, es decir, el intervalo
            en el que se calcula la integral y
            el número de pasos del algortimo
  SALIDAS: en este ejemplo, la temperatura media
            de la habitación
  VARIABLES:
            t1, t2, h, suma, ti, tempi, tempm: REAL
            i, n: ENTERO
```

```
*/
#include <stdio.h>
#include <math.h>
main()
{  /* INICIO */
 float t1, t2, h, suma, ti, tempi, tempm;
 int i, n;

 puts("Introduce t1: ");
 scanf("%f", &t1);
 puts("Introduce t2: ");
 scanf("%f", &t2);

 puts("Introduce numero de pasos de integracion: ");
 scanf("%d", &n);

 suma = 0.0;
 h = (t2-t1)/n;

 for(i=0; i<n; i++)
 {
       ti = t1+h*i;
       tempi = 50.0*(exp(-ti)+0.1*sin(10*ti));
       suma += tempi*h;
 }

 tempm = suma/(t2-t1);
 printf("\n\nLa temperatura media es: %f\n", tempm);

} /* FIN */
```

7.7 Para saber más

Existen multitud de libros en C ya que es uno de los lenguajes de programación más populares y extendidos que existen. El presente capítulo no ha pretendido dar una explicación detallada y precisa del lenguaje, sino explicar de forma sencilla los elementos básicos. Si se va a programar en

lenguaje C, es imprescindible "beber de las fuentes" y hacerse con el libro sobre C por antonomasia: "El lenguaje de programación C. Brian W. Kernighan, Dennis M. Ritchie. Prentice-Hall". Otro título interesante, debido a que se centra en la utilización del lenguaje desde Unix es "Curso de C bajo UNIX", Llanos D.R., Universidad de Valladolid, 1998.

7.8 Ejercicios propuestos

1. Elaborar un programa que solicite el valor del radio y devuelva por pantalla la longitud de la circunferencia y el radio del círculo.

2. Elaborar un programa que represente en una tabla por pantalla los valores de altura y presión correspondiente en alturas *h* desde cero a 80 Km. y tomadas cada 10 Km. Calcular la presión *P* según la ecuación: *P[=g/cm^2] = 1035*exp(-0.12*h[=Km])*

3. Elaborar un programa con un bucle for que presente en pantalla los números enteros entre 0 y 50.

4. Elaborar un programa con un bucle while que presente en pantalla los números enteros entre 0 y 50.

5. Elaborar un programa que solicite dos números enteros y plantee su producto como sumas sucesivas.

6. Elaborar un programa que solicite tres números enteros y compruebe si tomados dos a dos pueden sumar el tercero, avisando de la posible coincidencia.

7. Elaborar un programa que solicite los diez números enteros componentes de un vector de dimensión diez, y devuelva como resultado su suma.

8. Elaborar un programa que solicite los diez números enteros componentes de un vector de dimensión diez, y devuelva como resultado el mayor de todos ellos.

9. Elaborar un programa que solicite un número entero y devuelva como resultado la cantidad de veces que es divisible por dos.

10. Elaborar un programa de tipo sistema experto de botánica que mediante bucles if y en función de la longitud y el tipo (caduca o perenne) de las hojas de los árboles sea capaz de devolver el nombre del árbol para el que se responde a las preguntas planteadas.

11. Elaborar un programa de tipo sistema experto de control aéreo que mediante un bucle switch y en función de la tecla pulsada sea capaz de devolver el nombre completo del tipo de aeronave que se aproxima.

8. Matlab

8.1 Introducción

Una de las razones por las que Matlab se ha convertido en una herramienta tan popular dentro de las Ciencias e Ingenierías son

- Su capacidad de realizar operaciones con matrices de forma sencilla. Así podrán realizarse operaciones con matrices (sumas, restas, multiplicaciones, inversiones, etc.) sin necesidad de preocuparse por programar bucles que recorran los elementos de las matrices.
- Ni siquiera es necesario definir las dimensiones de las matrices utilizadas. Además estas dimensiones pueden alterarse si es necesario.
- Presenta una gran facilidad de crear gráficos sencillos, e interfaces de usuario.
- Existen un gran número de bibliotecas adicionales de funciones para facilitar el trabajo en muchas áreas de Ciencias e Ingenierías, denominadas *toolboxes*.

8.2 Comandos básicos de Matlab

La forma de definir una matriz no puede ser más sencilla: basta definir sus elementos entre corchetes. Por ejemplo, la matriz de dos filas y tres columnas:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

puede definirse en Matlab con las siguientes dos líneas de código (por conveniencia):

```
A=[1 2 3
   4 5 6]
```

Alternativamente la definición de la matriz se puede realizar de una forma más compacta, separando cada fila por un punto y coma, como en el siguiente ejemplo:

```
A=[1 2 3;  4 5 6]
```

Una vez definida la matriz podemos operar con ella. Así podemos evaluar su traspuesta:

Operación	Resultado
C=A'	C=[1 4 2 5 3 6]

Podemos multiplicarla por un escalar:

Operación	Resultado
C=3*A	C=[3 12 6 15 9 18]

Muchas veces se trabajará con matrices de un solo elemento (escalares), que pueden definirse prescindiendo de los corchetes:

```
a=1
```

Debe notarse que por defecto Matlab distingue entre mayúsculas y minúsculas, con lo que la matriz A es distinta de la matriz a.

En cuanto a los valores numéricos en que se pueden definir los elementos de la matriz, se pueden definir utilizando notación decimal convencional (punto fijo), y con coma flotante (factor de escala en potencia de 10), si se desea. Por ejemplo, es una definición válida:

Operación	Resultado
C=[-1.345 4.23E-1 2E-2 -Inf]	C = -1.3450 0.4230 0.2000 –Inf

Aunque se muestren en pantalla con menor número de cifras, los elementos de las matrices se almacenan en aritmética de punto flotante de doble precisión, incluyendo además valores para infinito (Inf), infinito negativo (-Inf) y valores inciertos (NaN). Las operaciones se realizan también en doble precisión. Si se desea visualizar mayor número de cifras significativas puede utilizarse la instrucción format long, que hace que a partir del momento en que se ejecuta los números aparezcan con 15 cifras significativas, en vez de las 5 por defecto:

Operación	Resultado
format long C=[-1.34578 4.23E4 2E-3 -Inf]	C = 1.0e+004 * -0.00013457800000 4.23000000000000 0.00000020000000 -Inf

Además de asignar valores concretos a los distintos elementos de las matrices, pueden generarse de forma automática valores para crear matrices de gran tamaño. Aparte de generar los elementos mediante lazos, frecuentemente se utilizan los comandos zeros, ones y logspace, y el operador de bucle implícito (:):

- El comando zeros(m,n) genera una matriz de m filas y n columnas con todos los elementos iguales a 0. Resulta muy útil para inicializar matrices que luego se van a rellenar de valores.

Operación	Resultado

`C=zeros(2,3)`	`C =`
	`0 0 0`
	`0 0 0`

- El comando ones(m,n) genera una matriz de m filas y n columnas con todos los elementos iguales a 1. Obsérvese que si se desea inicializar a otro valor, basta multiplicar por un escalar que sea el propio valor:

Operación	Resultado

`C=ones(2,3)`	`C =`
	`1 1 1`
	`1 1 1`

`C=5*ones(2,3)`	`C =`
	`5 5 5`
	`5 5 5`

- El operador de bucle implícito : permite generar vectores de valores sucesivos con un incremento constante. La utilización es inicio:incremento:final. Por defecto incremento vale 1, con lo que el comando se utilizaría como inicio:final

Operación	Resultado
`C=1:2:9`	`C =` `1 3 5 7 9`

- El comando logspace en cambio genera vectores espaciados logarítmicamente, muy útiles para generar barridos de frecuencia logarítmicos. La utilización es logspace(inicio, final, numero_puntos). Por defecto numero_puntos es 50.

Operación	Resultado
`C=logspace(1,3,3)`	`C =` `10 100 1000`

Hasta ahora hemos visto directamente en pantalla el resultado de todas las operaciones. Sin embargo esto puede no ser deseable cuando las matrices tienen muchos elementos. Si lo único que se desea es asignar el resultado de una operación a una variable, para luego seguir operando con ella, puede añadirse un punto y coma (;) final de la instrucción y el resultado no se mostrará en pantalla:

Operación	Resultado
`C=logspace(1,3,200);`	`C=` `<nada en pantalla, pero se asignan los valores a C >`

Para concluir esta parte mencionaremos otra gran ventaja de Matlab: Los elementos de las matrices pueden ser valores complejos, pudiendo operarse con estas matrices complejas de la misma forma que con las puramente reales.

Por ejemplo, puede definirse la matriz compleja $A = \begin{bmatrix} 3 \\ 5 \end{bmatrix} + \begin{bmatrix} 4 \\ 4 \end{bmatrix} j$ como:

Operación	Resultado

`C=[3+4i ; 5+4j]`	`C =`
	`3.0000 + 4.0000i`
	`5.0000 + 4.0000i`

Las matrices formadas por las partes reales e imaginarias pueden extraerse fácilmente con los comandos real e imag. El valor absoluto puede recuperarse con abs y el ángulo en radianes con angle.

Operación	Resultado

`C=[3+4i ; 5+4j];`
`Cr=real(C)`
`Ci=imag(C)`
`Cmod=abs(C)`
`Cangle=angle(C)`
	`Cr =`
	`3`
	`5`
	`Ci =`
	`4`
	`4`
	`Cmod =`
	`5.0000`
	`6.4031`
	`Cangle =`
	`0.9273`
	`0.6747`

8.3 Sistema de ayuda en Matlab

Como se ha podido comprobar, Matlab dispone de muchas funciones y operadores. Además, como se verá más adelante, añadir funciones nuevas es muy sencillo. Para facilitar el trabajo y no tener que recordar de memoria el nombre de todas las funciones disponibles Matlab dispone de un sistema de ayuda en línea de comandos bastante potente (aparte de la ayuda disponible en las últimas versiones en formato HTML o PDF en la ventana Help de la aplicación).

Si se desea conocer las operaciones que realiza una función determinada, cuáles son los argumentos de entrada y salida, y otras funciones parecidas basta pedir ayuda anteponiendo al nombre de la función la palabra help:

Operación	Resultado

`help logspace`	`LOGSPACE Logarithmically spaced vector.`
	`LOGSPACE(d1, d2) generates a row vector of 50 logarithmically equally spaced points between decades 10^d1 and 10^d2. If d2 is pi, then the points are between 10^d1 and pi.`
	`LOGSPACE(d1, d2, N) generates N points.`
	`See also LINSPACE, :.`

Si el problema es que no estamos seguros de qué función es la que hay que utilizar puede utilizarse el comando lookfor palabra_clave, que muestra en pantalla aquellas funciones que contienen palabra_clave en la primera línea de su descripción:

Operación	Resultado

`lookfor logarithm`	`LOGSPACE Logarithmically spaced vector.`
	`LOG Natural logarithm.`
	`LOG10 Common (base 10) logarithm.`
	`LOG2 Base 2 logarithm and dissect floating point number.`
	`BETALN Logarithm of beta function.`
	`GAMMALN Logarithm of gamma function.`
	`LOGM Matrix logarithm.`

Por último, si lo que se desea es comprobar qué funciones hay disponibles en alguna categoría puede ejecutarse el comando help sin argumentos:

Operación	Resultado
help	HELP topics: matlab\general - General purpose commands. matlab\ops - Operators and special characters. matlab\lang - Programming language constructs. matlab\elmat - Elementary matrices and matrix manipulation. matlab\elfun - Elementary math functions. matlab\specfun - Specialized math functions. matlab\matfun - Matrix functions - numerical linear algebra. ....

La ayuda de cualquiera de esas categorías puede obtenerse de forma similar a como se realizaba para una función cualquiera de Matlab. Por ejemplo, la lista de funciones en la categoría matlab\ops puede obtenerse con help ops:

Operación	Resultado
help ops	Operators and special characters. Arithmetic operators. plus - Plus + uplus - Unary plus + minus - Minus - uminus - Unary minus - mtimes - Matrix multiply * times - Array multiply .* mpower - Matrix power ^ ....

8.4 Operaciones básicas con matrices en Matlab

Matlab dispone de un buen número de operadores capaces de trabajar con matrices (o como caso particular, con escalares). Presentamos a continuación los más usuales.

Los operadores suma (+), resta (-) y multiplicación (*) de matrices tienen la interpretación usual en términos de operaciones matriciales

Operación	Resultado
D= [1 2 3]+[2 4 6]	D = 3 6 9

Operación	Resultado
D= [1 2 3]*[2; 4; 6]	D = 28

Un caso particular se presenta cuando uno de los operandos es una matriz y el otro es un escalar. En tal caso la operación se realiza sobre cada uno de los elementos de la matriz, aplicando el operador escalar elemento a elemento:

Operación	Resultado
D=[1 2 3]-2	D= -1 0 1

Operación	Resultado
D=3*[1 2 3]-2	D= 1 4 7

Si lo que se desea es multiplicar una matriz por ella misma un número de veces se dispone del operador exponenciación (^):

Operación	Resultado
D=[1 2; 3 4]^3	D = 37 54 81 118

Se dispone también de dos operadores para "división" de matrices, entendiendo por tal la solución de un sistema de ecuaciones. Los operadores son:

- División por la derecha (/): Si se calcula X=B/A, X será el vector solución del sistema de ecuaciones: $X * A = B$.

Operación	Resultado
D=[1 2] /[1 2; 3 4]	D = 1 0

- División por la izquierda (\): Si se calcula X=A\B, X será el vector solución del sistema de ecuaciones: $A * X = B$. Observar que, con escalares, esta división no es la usual: se divide el operando de la derecha *entre el de la izquierda.*

Operación	Resultado
D=[1 2; 3 4]\[1; 2]	D = 0 0.5000

En muchos casos, lo que se desea es realizar una operación determinada sobre cada elemento de una matriz. En tal caso es únicamente necesario anteponer al correspondiente operador el símbolo de punto (.) para que Matlab realice la correspondiente operación elemento a elemento.

Operación	Resultado
D=[1 2 3].*[1 2 3]	D= 1 4 9

Operación	Resultado
D=[1 2 3].^[1 2 3]	D= 1 4 27

Además de estos operadores básicos, se dispone de un gran número de funciones en Matlab para realizar operaciones más complejas. Además, como se verá posteriormente, es relativamente sencillo añadir funciones nuevas a las disponibles. Por ejemplo pueden evaluarse funciones trigonométricas con las funciones sin, cos, tan, atan (tangente inversa), sinh (seno hiperbólico), cosh, tanh, etc., de la siguiente forma:

Operación	Resultado
D=sin(pi/2)	D= 1

Observar que el argumento de las funciones trigonométricas se entiende que está en radianes y el número π está definido en la variable pi. Entonces, si se desea realizar operaciones con grados bastará multiplicar los grados por pi/180:

Operación	Resultado
D=sin(45*pi/180)	D= 0.7071

Es importante tener en cuenta que estos operadores pueden trabajar también con vectores, evaluándose la función en cada uno de los valores. Esto es muy

útil para generar señales. Por ejemplo 100 puntos de una señal sinusoidal de frecuencia 10 ciclos/seg se puede generar como:

Operación	Resultado
`t=1:100;` `D=sin(10*pi/180*t);`	

Por supuesto, existen muchos más operadores implementados en Matlab, que pueden verse pidiendo ayuda con help elfun (funciones matemáticas elementales), specfun (funciones matemáticas avanzadas), matfun (funciones matemáticas especiales para operar sobre matrices), datafun (análisis estadístico y espectral de señales), polyfun (funciones para operar con polinomios, que describiremos en el siguiente apartado), etc.

Destacaremos aquí únicamente la evaluación de logaritmos naturales (log), logaritmos decimales (log10), y la exponenciación (exp):

Operación	Resultado

`D= log10(100)`
`E= log(100)`
`F= exp(1)`
	`D =`
	`2`
	`E =`
	`4.6052`
	`F =`
	`2.7183`

Para concluir esta sección debemos mencionar que las operaciones sobre matrices se pueden aplicar también a matrices complejas, con la misma interpretación.

8.5 Operaciones básicas con polinomios en Matlab

Para trabajar con un polinomio en Matlab basta con crear un vector que contenga sus coeficientes en potencias decrecientes de la variable independiente. Así el polinomio $g(x) = x^2 + 2x - 0.5$ se crearía de la siguiente forma:

Operación	Resultado
`g=[1 2 -0.5]`	`g=` `1.0000 2.0000 -0.5000`

Una vez creado el vector asociado al polinomio, existen comandos para realizar distintas operaciones: puede calcularse el valor del polinomio en un punto (polyval), evaluar sus raíces de forma numérica (roots) o calcular su derivada respecto a la variable independiente (polyder):

Operación	Resultado
`valor=polyval(g,1)`	`valor =` `2.5000`

`r=roots(g)`	`r =`
	`-2.2247`
	`0.2247`

`h=polyder(g)`	`h =` `2 2`

Además pueden realizarse operaciones básicas entre polinomios: multiplicación de polinomios (conv) y división de polinomios (deconv)

Operación	Resultado
`h=[1 2]`	`h=` `1.0000 2.0000`
`k=conv(g,h)`	`k =` `2 6 3 -1`

Un caso importante se presenta cuando tenemos un cociente de polinomios en una variable (representación que se utiliza, por ejemplo, en teoría de sistemas, en procesado de señales, en automática, etc.). En tal caso, numerador y denominador se mantienen como estructuras independientes cuyo cociente forma un sistema, pudiendo realizarse, por ejemplo, una descomposición en fracciones parciales utilizando el comando residue. Este comando devuelve como resultado vectores columna con los *residuos* (los numeradores), los *polos* (las raíces del denominador) y el término directo de la descomposición. Por ejemplo:

$$\frac{x^2-2x+1}{x^2+3x+2}=\frac{-9}{x+2}+\frac{4}{x+1}+1$$

Operación	Resultado

`[residuos,polos,directo]=residue([1 -2 1],[1 3 2])`	`residuos =`
	`-9`
	`4`
	`polos =`
	`-2`
	`-1`
	`directo =`
	`1`

Cuando aparecen raíces del denominador repetidas, se generan los residuos para potencias crecientes del monomio correspondiente al polo. Por ejemplo:

$$\frac{x^2+3x+1}{x^2+2x+1}=\frac{1}{x+1}+\frac{-1}{(x+1)^2}+1$$

Operación	Resultado

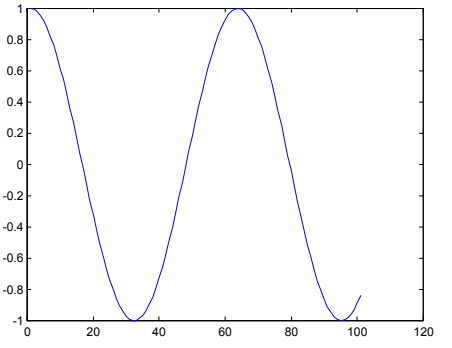
`[residuos,polos,directo]=residue([1 3 1],[1 2 1])`	`residuos =`
	`1`
	`-1`
	`polos =`
	`-1`
	`-1`
	`directo =`
	`1`

8.6 Representaciones gráficas en Matlab

Hasta ahora hemos visto cómo se pueden realizar operaciones con Matlab. Ahora vamos a ver cómo pueden utilizarse las capacidades gráficas que proporciona Matlab para visualizar los resultados y poder analizarlos de una forma sencilla.

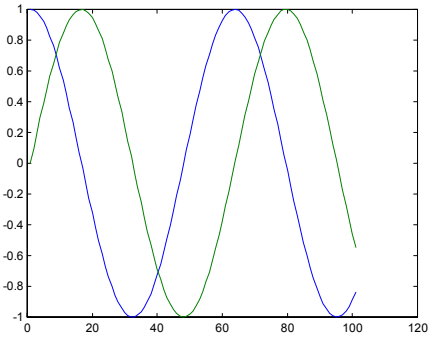
El comando básico para representaciones gráficas en Matlab es plot. La forma de utilizarlo no puede ser más sencilla: Si lo que se desea es dibujar un vector basta incluir como argumento entre paréntesis el vector a representar. Se abre entonces una ventana que dibuja el vector: en abscisas se indica el índice correspondiente del elemento de la matriz y en ordenadas su valor.

Operación	Resultado

`t=0:0.1:10;`
`D=cos(t);`
`plot(D)`
	

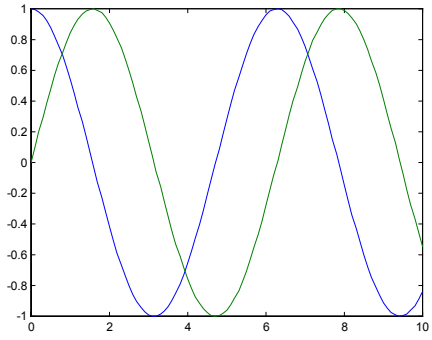
En el caso en que se desee representar varios vectores almacenados en una misma matriz, es necesario tener en cuenta que se representa cada *columna* de la matriz en un color. Por lo tanto hay que asegurarse que las variables a representar aparezcan como columnas. Para ello puede ser necesario trasponer la correspondiente matriz:

Operación	Resultado

`t=0:0.1:10;`
`D=[cos(t);`
`sin(t)];`
`plot(D')`
	

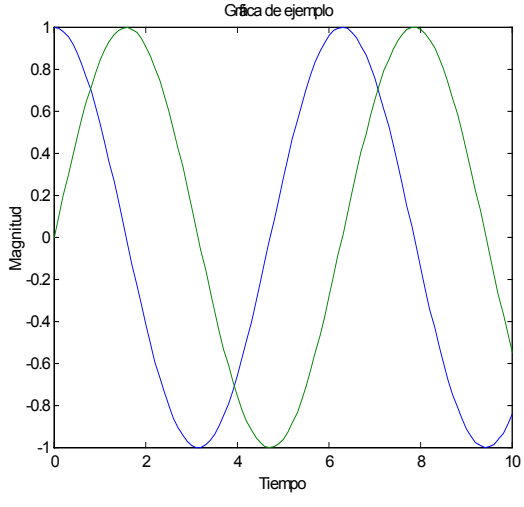
Generalmente se desea que el eje de abscisas aparezca en una determinada escala. Para ello puede añadirse como primer argumento un vector que contenga los valores que corresponden a cada índice de la matriz:

Operación	Resultado

`t=0:0.1:10;`
`D=[cos(t); sin(t)];`
`plot(t,D')`
	

Una vez realizada la correspondiente gráfica pueden añadirse títulos (title) y etiquetas en los ejes X (xlabel) e Y (ylabel):

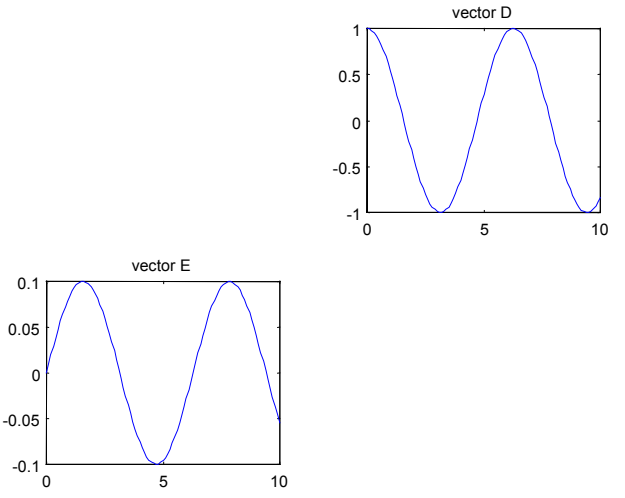
Operación	Resultado

`t=0:0.1:10;`
`D=[cos(t); sin(t)];`
`plot(t,D')`
`title('Gráfica de ejemplo')`
`xlabel('Tiempo')`
`ylabel('Magnitud')`
	

A veces las representaciones gráficas se visualizan mejor en escala logarítmica (generalmente cuando hay valores de magnitudes de órdenes muy diferentes). Para ello en Matlab se dispone de las funciones semilogx (escala logarítmica en X y lineal en Y), semilogy (escala lineal en X y logarítmica en Y) y loglog (escala doblemente logarítmica).

Otro comando útil para representaciones gráfica en Matlab es subplot. Con este comando en una misma ventana pueden representarse varias gráficas con escalas diferentes. La forma de operar es la siguiente: con subplot(m,n,i) la ventana quedaría dividida en m particiones horizontales y n verticales; la siguiente gráfica se realizaría en la partición número i, contándose consecutivamente desde la primera fila:

Operación	Resultado

t=0:0.1:10;
D=cos(t);
E=sin(t);
subplot(2,2,2);
plot(t,D)
title('vector D')
subplot(2,2,3);
plot(t,E)
title('vector E')
	

Otro comando útil a la hora de realizar representaciones gráficas es hold. Por defecto, cuando se realiza una gráfica se destruye la anterior que hubiera en la ventana. Una vez activado el comando con hold on se superpondrán las gráficas en la ventana actual, modificando la escala de los ejes si fuera necesario. Para desactivar la superposición puede utilizarse hold off. Si se ha seleccionado una partición de la ventana con subplot, el funcionamiento es similar, manteniendo lo que estuviera en la partición correspondiente, debiendo ejecutarse el comando hold on por cada partición a mantener.

Una vez visto cómo pueden generarse representaciones gráficas en Matlab vamos a ver cómo podemos tener diferentes ventanas gráficas. Para ello se utiliza el comando figure. Como tal, el comando figure sin argumentos crea una ventana nueva, que pasa a ser la activa (aquella donde se aplicarán los siguientes comandos gráficos). A cada ventana que se crea se le asigna un número (de forma sucesiva, comenzando en 1), de manera que puede seleccionarse como ventana activa cualquier otra ventana previamente creada añadiendo el número de ventana como argumento: figure(Numero_de_ventana); si no existiera una ventana con ese número, se crearía.

8.7 Programación en Matlab

Hasta ahora se ha mostrado cómo puede utilizarse Matlab de forma interactiva desde la línea de comandos. Sin embargo, la potencia de Matlab se consigue con la facilidad que presenta para escribir y depurar programas que realicen determinadas tareas.

Presentaremos a continuación las Estructuras de selección y repetición que suelen utilizarse dentro de los programas de Matlab para implementar los

algoritmos, para luego describir la forma de realizar programas en Matlab, en base a *ficheros de comandos* y *funciones*

8.7.1 Estructuras de selección

Muchas veces se desea ejecutar un fragmento de código únicamente si se cumple una determinada condición. Para ello puede utilizarse la instrucción if...else..., con la siguiente estructura:

```
if condicion
   instrucciones1
else
   instrucciones2
end
```

Si condicion es cierto se ejecuta el bloque de instrucciones instrucciones1; si no se ejecuta el bloque de instrucciones instrucciones2, tal y como se vio en el capítulo de Algorítmica.

La forma más sencilla de escribir una condición en Matlab será comparando una variable con otra variable o una constante. Para ello pueden utilizarse los siguientes operadores de comparación:

Igual que	==
Distinto que	~=
Menor que	<
Mayor que	>
Menor o igual que	<=
Mayor o igual que	>=

Se debe observar que la comparación de dos cantidades se escribe con dos símbolos de igualdad consecutivos. Para generar la tilde en la comparación de desigualdad en aquellos teclados en los que no se dispone del carácter tilde, puede generarse manteniendo pulsada la tecla Alt y escribiendo 126 en el teclado numérico de la derecha.

Por supuesto la condición a comprobar puede ser tan complicada como se desee: varias comparaciones pueden combinarse utilizando los operadores lógicos AND (&), OR (|) y NOT (~).

Volviendo a la instrucción if, debe tenerse en cuenta que tanto instrucciones1 como instrucciones2 pueden contener otras instrucciones if, pero para evitar que la estructura se complique y facilitar la depuración de los programas se puede utilizar la estructura if...elseif...else..., con la siguiente estructura:

```
if condicion
   instrucciones1
elseif condicion2
   instrucciones2
elseif condicion3
   instrucciones3
else
   instrucciones4
end
```

El funcionamiento es el siguiente: se comprueba primero condicion. Si es cierta se ejecuta instrucciones1; si no, se comprueba condicion2. Si condicion2 es cierta se ejecuta instrucciones2; si no, se comprueba condicion3, que si es cierta se ejecuta instrucciones3. Finalmente si ninguna de las condiciones es cierta se ejecuta instrucciones4. Observar que en el momento en que se cumple una de las condiciones ya no se verifican el resto de las condiciones.

Para estructuras de selección complejas puede utilizar la instrucción switch, que resulta especialmente útil para ejecutar instrucciones diferentes según el valor de alguna variable. La estructura es:

```
switch variable
case valor_a
   instrucciones1
case valor_b
   instrucciones2
case valor_c
   instrucciones3
otherwise
   instrucciones4
end
```

La instrucción funciona de esta forma: se comprueba primero si la variable tiene el valor valor_a. Si es así se ejecuta el conjunto de instrucciones instrucciones1; si no, se comprueba si tiene el valor valor_b. Si tiene este valor se ejecuta instrucciones2, y así sucesivamente. Si no toma ninguno de los valores especificados se ejecuta instrucciones4. Observar que en el

momento en que se verifica que la variable toma alguno de los valores, ya no se comprueban el resto de los valores.

8.7.2 Estructuras de repetición

A la hora de programar algoritmos suele ser necesario realizar lazos, tal y como se vio en el capítulo de Algorítmica. Para ello Matlab dispone de las funciones for y while. La estructura del bucle determinista for es:

```
for variable=inicio:incremento:fin,
     instrucciones_a_ejecutar;
end
```

En el caso en que se desee ejecutar el bucle mientras una condición de parada sea cierta puede utilizarse el bucle condicional while:

```
while condicion,
   instrucciones_a_ejecutar;
end
```

Observar que no existe un comando para comprobar la condición sólo en el final del bucle (no existe un comando equivalente al REPETIR...HASTA que se vio en algorítmica). Si se desea comprobar al final del bucle, puede realizarse moviendo la condición al comienzo del bucle:

```
variable_condicion=1;
while variable_condicion==1,
   instrucciones_a_ejecutar;
   variable_condicion=condicion;
end
```

8.7.3 Ficheros de comandos

La forma más sencilla de escribir un programa en Matlab es generando un fichero de comandos. Para ello, basta escribir un fichero de texto con las instrucciones que escribiríamos en la línea de comandos y grabarlo con extensión .m. en un directorio que sea accesible a Matlab. Para ejecutarlo basta escribir su nombre en la línea de comandos, ejecutándose las líneas como si se escribieran en la línea de comandos. La única diferencia es que los comandos que se ejecutan no aparecen en pantalla, salvo que previamente se ejecute echo on.

Hemos dicho que los ficheros a ejecutar deben estar en directorios accesibles a Matlab: pero ¿cuáles son esos directorios? Aparte del directorio de trabajo actual (que puede visualizarse con cd y cambiarse con cd nuevo_directorio),

Matlab guarda en memoria las trayectorias de aquellos directorios que le indiquemos, donde haya ficheros de Matlab. Normalmente estas trayectorias se definen en el fichero de configuración matlabrc.m, pero se pueden cambiar durante la ejecución. Para ello puede utilizarse la opción de menú File->Set Path o utilizar el comando path en la línea de comandos.

Al ejecutar cualquier comando, Matlab consulta primero si forma parte de las funciones predefinidas en Matlab, luego si está en el directorio actual y finalmente si existe en alguno de los directorios de la trayectoria en path. Muchos errores ocurren por existir ficheros con el mismo nombre en varios directorios. Para aclarar cuál es el comando que se ejecuta puede consultarse con which.

Operación	Resultado
`which residue`	`D:\programas\matlab\toolbox\matlab\polyfun\residue.m`

8.7.4 Funciones

Si el fichero de comandos de Matlab realiza unas acciones concretas sobre unas variables de entrada para generar unas variables de salida, es mejor convertirlo en una función de Matlab. De esta forma, el resto de las variables que se utilicen en la función serán variables locales, con lo que no alterarán el contenido de las variables definidas en el entorno de comandos de Matlab. Por supuesto, las funciones pueden llamarse también desde otros ficheros de comandos o funciones de Matlab. Para que un fichero de comandos pueda considerarse una función de Matlab, basta crear un fichero con nombre nombre_funcion.m y cuya primera línea del fichero (excluyendo comentarios) tenga esta estructura:

```
[variable1_salida,variable2_salida]=nombre_funcion(variable1_entrada,variable2_entrada)
```

Por supuesto al definir la función pueden tenerse tantas variables de entrada y de salida como se quiera. Pero además, para que las funciones sean lo más versátiles posibles, a la hora de utilizar la función, se pueden recoger sólo aquellas variables de entrada que necesitemos, y pasarle menos argumentos de los que estén declarados en la función. Esto último es muy útil para utilizar parámetros a los que se les pueda asignar valores por defecto dentro de la función si no se pasa la variable correspondiente. Así si escribimos un fichero potencias.m, en algún directorio accesible por Matlab, con el siguiente contenido:

```
% POTENCIAS calcula potencias enteras
%
% POTENCIAS(x) genera las potencias segunda, tercera y cuarta
de x
%
% Por defecto x=2
%
function [x2,x3,x4]=potencias(x)


if (nargin==0)
   x2=2;
   x3=4;
   x4=8;
else
   x2=x*x;
   x3=x2*x;
   x4=x3*x;
end
```

Si se pasa un argumento numérico se calculan algunas de sus potencias. Si no se pasa ninguno, se asignan las potencias de 2:

Operación	Resultado

`[cuadrado,cubo]=prueba(3)`	`cuadrado =`
	`9`
	`cubo =`
	`27`

`[cuadrado,cubo,cuarta]=prueba`	`cuadrado =`
	`2`
	`cubo =`
	`4`
	`cuarta =`
	`8`

Este mismo ejemplo nos sirve para comentar la creación de ayudas en las funciones: Matlab entiende como descripción de la función todas las líneas de comentario que aparecen en la cabecera antes de la primera línea interpretable (en el caso de una función la que contiene la palabra function). Al pedir ayuda

del comando, mostrará todas esas líneas de ayuda. Si la búsqueda se realiza con lookfor se mostrará sólo la primera línea.

Operación	Resultado
help potencias	POTENCIAS calcula potencias enteras POTENCIAS(x) genera las potencias 2, 3 y 4 de x Por defecto x=2
lookfor potencias	POTENCIAS calcula potencias enteras

Para facilitar la labor al usuario futuro de las funciones que creemos, es buena política de programación el comenzar los ficheros de Matlab por unas líneas de comentarios que describan lo que hace la función, y sus parámetros de entrada y salida. La primera línea de comentarios debería describir muy brevemente para qué se utiliza la función. Con esto se facilitan enormemente las búsquedas de funciones en Matlab.

8.8 Codificación de casos típicos

En este apartado realizaremos la codificación en Matlab de los casos típicos presentados en el apartado 6.2. La explicación de la resolución puede encontrarse en dicho apartado, aquí simplemente plantearemos la solución y, en algún caso, haremos alguna consideración adicional.

8.8.1 Sumatorios y medias aritméticas. Centro de masas

```
% CENTROMASAS calcula el centro de masas de N partículas

% ALGORITMO centro de masas
% ENTRADA: número de partículas y datos de cada partícula
% SALIDA:  la masa media y el centro de masas
% VARIABLES: como en MATLAB no hace falta declarar las
% variables solo ponemos los nombres para hacerlo lo mas
% parecido posible al algoritmo que se vio en dicho
% capitulo
%          k,i: ENTEROS
```

```
%    MT, MX, MY, MZ, MM: REAL
%          C(k,4): matriz de REAL


% Leer los datos por teclado
k = input('Numero de particulas: ');

% Ahora se van leyendo los datos de cada una de las particulas
for i=1:1:k
   C(i,1) = input('Masa de la particula: ');
   C(i,2) = input('Coordenada x: ');
   C(i,3) = input('Coordenada y: ');
   C(i,4) = input('Coordenada z: ');
end

%Inicializacion de las otras variables
MT=0.0; MX=0.0; MY=0.0; MZ=0.0;

% Bucle para calcular el centro de masas
for i=1:1:k
   MT = MT + C(i,1);
   MX = MX + C(i,1)*C(i,2);
   MY = MY + C(i,1)*C(i,3);
   MZ = MZ + C(i,1)*C(i,4);
end

% Calculos finales
MT = MT/k;
MX = MX/MT;
MY = MY/MT;
MZ = MZ/MT;

% La forma mas sencilla de escribir en Matlab el valor de
% una variable es poner directamente el nombre de ella:
MT
```

```
MX
MY
MZ


% Si queremos escribir los valores con algun comentario de
texto:
s=['Masa media ' num2str(MT)];
display(s);


s=['Coordenada x del centro de masas ' num2str(MX)];
display(s);


s=['Coordenada y del centro de masas ' num2str(MY)];
display(s);


s=['Coordenada x del centro de masas ' num2str(MZ)];
display(s);
```

8.8.2 Resolución de ecuaciones no lineales. Método de Newton-Raphson

Escribimos las funciones que vamos a utilizar en ficheros aparte (funcionf.m y funcionfder.m):

```
% FUNCIONF Ecuación no lineal a resolver por Newton-Raphson
function [fx] =  funcionf(x)
fx = x^2+0.5-exp(-x);
```

```
% FUNCIONFDER Derivada de la ecuación no lineal a resolver por
     Newton-Raphson
function [dx] =  funcionfder(x)
dx = 2*x+exp(-x);
```

```
% NEWTON Resolución de Ecuaciones no lineales por Newton-
Raphson


% Algoritmo Newton Raphson
% Leer variables por teclado
```

```
x1 = input('Aproximación inicial de la solución: ');
cota = input('Error que se quiere obtener: ');
N = input('Numero de iteraciones: ');

%     Inicializar variables
i = 1;
error = Inf;

%     Bucle del programa
while (i<=N) & (error > cota)
   fx1 = funcionf(x1);
   fx1der = funcionfder(x1);
   x2 = x1-fx1/fx1der;
   error = abs(x2-x1);
   i = i+1;
   x1 = x2;
end

['Solucion = ' num2str(x2)]
['Numero de iteraciones = ' num2str(i-1)]
['Valor final del error = ' num2str(error)]
i = i-1;
if (i<=N) | (error <cota)
   display('Se ha alcanzado la solucion correcta');
end
```

A continuación se presenta una versión mejorada del algoritmo de Newton Raphson, que utiliza las capacidades gráficas de Matlab, para verificar el funcionamiento del algoritmo:

```
% NEWTONG Resolución de Ecuaciones no lineales por Newton-
Raphson (Versión Gráfica)
%
% Algoritmo Newton Raphson dibujando en la pantalla


%Definimos un eje de tiempos para los dibujos
t=-5:0.1:5;
```

```
y=t.^2+0.5-exp(-x);

% Leer variables por teclado
x1 = input('Aproximación inicial de la solución: ');
cota = input('Error que se quiere obtener: ');
N = input('Numero de iteraciones: ');

%     Inicializar variables
i = 1;
error = Inf;

%     Bucle del programa
while (i<=N) & (error > cota)
   fx1 = funcionf(x1);
   fx1der = funcionfder(x1);
   x2 = x1-fx1/fx1der;

   %Desde matlab podemos llamar a cualquier otro fichero
        %de comandos
   dibujar
   error = abs(x2-x1);
   i = i+1;
   x1 = x2;
end
['Solucion = ' num2str(x2)]
['Numero de iteraciones = ' num2str(i-1)]
['Valor final del error = ' num2str(error)]
i = i-1;
if (i<=N) & (error <cota)
   display('Se ha alcanzado la solucion correcta');
else
   display ('No se ha alcanzado la solucion correcta ');
end
```

La función dibujar.m es utilizada por el fichero de comandos anterior para representar la función no-lineal:

```
% DIBUJAR Dibuja la ecuación no lineal a resolver por Newton-
Raphson


%Dibujar x1,x2
fx1 = funcionf(x1);
fx2 = funcionf(x2);
tmin = -5;
tmax = 5;

ftmin = (fx1-fx2)/(x1-x2)*(tmin-x1)+fx1;
ftmax = (fx1-fx2)/(x1-x2)*(tmax-x1)+fx1;

plot(t,y)
hold on
%dibujamos un eje horizontal
plot([tmin tmax], [0.0 0.0], 'k');
%dibujamos la linea
plot([tmin tmax], [ftmin ftmax], 'g')
title(['Solucion = ' num2str(x2)])
hold off
display('Pulsa una tecla para seguir!!!!!!');
pause
```

8.8.3 Operaciones con vectores. Producto escalar y vectorial

```
% PRODUCTO Calcula el producto escalar y vectorial de dos
  vectores
%
% ALGORITMO: producto escalar y vectorial de dos vectores
% ENTRADAS: dos vectores
% SALIDAS: el producto escalar, vectorial y el angulo que
% forman ambos
% VARIABLES: i: ENTERO
%           v1(3), v2(3), productov(3): VECTORES REALES
%           productoe, modulov1, modulov2, angulo:REALES
```

```
% INICIO
% Inicializacion de variables
productoe =  0.0;
modulov1 =  0.0; % No haría falta
modulov2  = 0.0; % No haría falta

% Lectura de los elementos de los vectores v1 y v2
% cada vector lo leemos de una forma diferente

% Lectura de v1
v1 = input('Introduce el vector v1 (pon todo el vector:[
   ]):');

%Lectura de v2
for i=1:1:3;
   v2(i) =  input(['Elemento ' num2str(i) ' de v2: ']);
end

% Calculamos el modulo de v1
modulov1 = sqrt(sum(v1.*v1));
% Calculamos el modulo de v2
modulov2 = sqrt(sum(v2.*v2));

% Producto escalar
productoe = sum(v1.*v2);

% Angulo que forman los dos vectores
angulo = acos(productoe/(modulov1*modulov2));

% Producto vectorial
productov(1) = v1(2)*v2(3) - v1(3)*v2(2);
productov(2) = v1(3)*v2(1) - v1(1)*v2(3);
productov(3) = v1(1)*v2(2) - v1(2)*v2(1);

% Presentacion de resultados al usuario
```

```
display(['Dados ' num2str(v1) ' y ' num2str(v2) 'los resultados
  son: ']);
display(['Producto escalar: ' num2str(productoe) ]);
display(['Modulo vector v1: ' num2str(modulov1)]);
display(['Modulo vector v2: ' num2str(modulov2)]);
display(['Angulo que forman los dos vectores: '
  num2str(angulo*180/pi)]);
display(['Producto vectorial: '  num2str(productov)]);


% Tambien podemos dibujar los dos vectores y el producto
  vectorial
plot3([0 v1(1)],[0 v1(2)], [0 v1(3)],'b')
hold  on
plot3([0 v2(1)],[0 v2(2)], [0 v2(3)],'g')
plot3([0 productov(1)],[0 productov(2)], [0 productov(3)],'r')
hold  off


% FIN
```

8.8.4 Resolución de sistemas de ecuaciones lineales. Regla de Cramer

Para la regla de Cramer aplicada al caso de un sistema 3x3, primero programamos una función que calcula el determinante:

```
% DET_TRES Calcula el determinante de una matriz 3x3
% funcion DET_TRES
% ENTRADAS: una matriz 3x3
% SALIDAS: el determinante de la matriz


function [y] = det_tres(a)
y = a(1,1)*a(2,2)*a(3,3)+a(1,2)*a(2,3)*a(3,1)+...
   a(2,1)*a(3,2)*a(1,3)-a(3,1)*a(2,2)*a(1,3)-...
   a(2,1)*a(1,2)*a(3,3)-a(3,2)*a(2,3)*a(1,1);
```

y el programa principal que aplica la regla de Cramer al sistema de ecuaciones que se introduzca por teclado:

```
% CRAMER Calcula el determinante de una matriz 3x3 por la regla
  de Cramer
%
% ALGORITMO regla de Cramer
% ENTRADAS: matriz A y vector b
% SALIDA: la solucion del sistema de ecuaciones
% VARIABLES: A(3,3), b(3), d(3,3), deta, detd,x: REALES
%           i,j,k: ENTERO

% Se introducen los datos de la matriz A
for i=1:1:3
   for j=1:1:3
      A(i,j) = input(['Introduce el elemento de la fila '
                    num2str(i) ' y columna ' num2str(j) ...
                    ' de la matriz A: ']);
   end
end

for i=1:1:3
   b(i) = input(['Introduce el elemento ' num2str(i) ' de
   b: ']);
end

% Se pasa a calcular el determinante de A
deta = DET_TRES(A);

if(deta ~= 0)
   for i=1:1:3
      for j=1:1:3
         for k=1:1:3
            if i==j
               d(k,j) = b(k);
            else
               d(k,j) = A(k,j);
            end % if
```

```
         end % for k
      end % j
      detd = DET_TRES(d)
      x(i) = detd/deta;
   end % i
   display('La solucion del sistema es: ');
   x
else
   display('El determinante de A es cero, asi que el sistema no
  tiene solucion');
end % if
```

Como se vio en la sección 8.4, este ejercicio se puede resolver de forma mucho más sencilla y además para un caso general, utilizando el operador \ de Matlab.

8.8.5 Resolución de una integral definida

La ventaja de programar este problema en MATLAB es que podemos introducir fácilmente gráficos. Veamos primero una versión que no tiene gráficos:

```
% INTDEF Calcula una integral definida
%
% ALGORITMO: resolucion de una integral definida
  % ENTRADAS: intervalo temporal, numero de pasos del
% algoritmo de integracion
  % SALIDAS: la temperatura media
  % VARIABLES: i, n: ENTERO
  %              t1, t2, h, suma, ti, tempi, tempm: REALES

% INICIO

% Leemos el instante inicial, el final y el numero de pasos
t1 = input('Instante inicial ');
t2 = input('Instante final ');
n = input('Numero de pasos del algoritmo ');

% Inicializacion de variables
```

```
suma = 0.0;
% Duracion de cada subintervalo
h = (t2-t1)/n;

for i=0:1:n-1;
   ti = t1+h*i;
   tempi = 50.0 * (exp(-ti)+0.1*sin(10.0*ti));
   suma = suma + tempi*h;
end
tempm  = suma/(t2-t1);

% Se presentan los resultados al usuario
display(['La temperatura media de la habitacion es: '
  num2str(tempm)]);

% FIN
```

La versión con representación gráfica:

```
% INTDEFG Calcula una integral definida (versión gráfica)
%
% ALGORITMO: resolucion de una integral definida
% ENTRADAS: intervalo temporal, numero de pasos
% SALIDAS: la temperatura media y un grafico con la
  aproximacion
% VARIABLES: i, n: ENTERO
%  t1, t2, h, suma, ti, tempi, tempm: REALES
% t, y: REALES

% INICIO

% Leemos el instsnte inicial, el final y el numero de pasos
t1 = input('Instante inicial ');
t2 = input('Instante final ');
n = input('Numero de pasos del algoritmo ');

% Inicializacion de variables
```

```
suma = 0.0;
% Duracion de cada subintervalo
h = (t2-t1)/(n-1);

y = [];
for i=0:1:n-1;
   ti = t1+h*i;
   tempi = 50.0 * (exp(-ti)+0.1*sin(10.0*ti));
   suma = suma + tempi*h;
   y = [y tempi];
end
tempm  = suma/(t2-t1);

% Se presentan los resultados al usuario
display(['La temperatura media de la habitacion es: '
  num2str(tempm)]);

% Se dibujan los resultados
ty=t1:h:t2;
bar(ty,y,1,'w')
hold on
t=t1:(t2-t1)/100:t2;
x=50.0 * (exp(-t)+0.1*sin(10.0*t));
plot(t,x,'bo')
hold off
title(['La temperatura media de la habitacion es: '
  num2str(tempm)]);
% FIN
```

Si queremos aplicar el método de los rectángulos a otra función diferente:

$$f(x) = \mathrm{sinc}^2(t) = \left(\frac{\sin(\pi t)}{\pi t}\right)^2$$

el programa en MATLAB sería:

```
% INTDEFG Calcula una integral definida (versión gráfica)
%
% ALGORITMO: resolucion de una integral definida
% ENTRADAS: intervalo temporal, numero de pasos
% SALIDAS: valor de la integral y un grafico con la
%         aproximacion
% VARIABLES: i, n: ENTERO

%     t1, t2, h, suma, ti, tempi, tempm: REALES
%     t, y: REALES
% INICIO
% Leemos el instsnte inicial, el final y el numero de pasos
t1 = input('Instante inicial ');
t2 = input('Instante final ');
n = input('Numero de pasos del algoritmo ');

% Inicializacion de variables
suma = 0.0;
% Duracion de cada subintervalo
h = (t2-t1)/(n-1);

y = [];
for i=0:1:n-1;
   ti = t1+h*i;
   tempi = 5.0 * (sinc(ti).*sinc(ti));
   suma = suma + tempi*h;
   y = [y tempi];
end

% Se presentan los resultados al usuario
display(['El valor de la integral es: ' num2str(tempm)]);

% Se dibujan los resultados
ty=t1:h:t2;
bar(ty,y,1,'w')
```

```
hold on
t=t1:(t2-t1)/100:t2;
x=5.0 * (sinc(t).*sinc(t));
plot(t,x,'b')
v=axis;
axis([t1  t2 v(3) v(4)])
hold off
title(['El valor de la integral es: ' num2str(tempm)]);
% FIN
```

8.9 Para saber más

Aparte de los manuales de Matlab, existe un gran número de libros publicados sobre la utilización de Matlab para la resolución de problemas en distintas áreas. En particular mencionaremos los siguientes libros:

- Biran, A., Breiner, M., *Matlab for Engineers*, Addison-Wesley, 1997
- Díez-Ruano, J.L., Prieto-Mahon, R., *Procesamiento de Señales. Prácticas con Matlab y Simulink*, Univ. Pol. Valencia, 1998
- Etter, D. M., *Solución de problemas de ingeniería con Matlab*, Prentice-Hall, 1997
- Nakamura, S., *Análisis Numérico y Visualización Gráfica con Matlab*, Prentice-Hall, 1997
- Ogata, K., *Solving Control Engineering Problems using Matlab*, Prentice-Hall, 1994
- Pérez López, C., *Matemática Informatizada con Matlab*, Ra-Ma, 1996
- Quintela Estévez, P., *Introducción a Matlab y sus aplicaciones*, Univ. Santiago de Compostela, 1997

Es también útil comprobar las páginas de los creadores de Matlab (www.mathworks.com) para mantenerse al día de nuevas versiones de Matlab o toolboxes, así como de los comentarios técnicos.

Finalmente mencionar que existen una serie de documentos de introducción a las diferentes versiones de Matlab que pueden descargarse desde la siguiente dirección: http://fcapra.ceit.es/AyudaInf.

Apéndice A. Tabla Código ASCII estándar

	0	1	2	3	4	5	6	7	8	9
0	nul	soh	stx	etx	eot	enq	ack	bel	bs	ht
1	nl	vt	np	cr	so	si	sle	dc1	dc2	dc3
2	dc4	nak	syn	etb	can	em	sub	esc	fs	gs
3	rs	us	sp	!	"	#	$	%	&	'
4	(	)	*	+	,	-	.	/	0	1
5	2	3	4	5	6	7	8	9	:	;
6	<	=	>	?	@	A	B	C	D	E
7	F	G	H	I	J	K	L	M	N	O
8	P	Q	R	S	T	U	V	W	X	Y
9	Z	[	\	]	^	_	`	a	b	c
10	d	e	f	g	h	i	j	k	l	m
11	n	o	p	q	r	s	t	u	v	w
12	x	y	z	{	|	}	~	del		

La Tabla se utiliza de la siguiente forma. La primera cifra (las dos primeras cifras, en el caso de los números mayores o iguales que 100) del número ASCII correspondiente a un carácter determinado figura en la primera columna de la Tabla, y la última cifra en la primera fila de la Tabla. Sabiendo la fila y la columna en la que está un determinado carácter puede componerse el número correspondiente. Por ejemplo, la letra A está en la fila 6 y la columna 5. Su número ASCII es por tanto el 65. El carácter % está en la fila 3 y la columna 7, por lo que su representación ASCII será el 37. Obsérvese que el código ASCII asocia números consecutivos con las letras mayúsculas y minúsculas ordenadas alfabéticamente. Esto simplifica notablemente ciertas operaciones de ordenación alfabética de nombres cuando se utilizan comparaciones a la hora de programar.

En la Tabla aparecen muchos caracteres no imprimibles (todos aquellos que se expresan con 2 ó 3 letras). Por ejemplo, el **ht** es el tabulador horizontal, el **nl** es el carácter *nueva líne*a, etc.

Apéndice B. Problemas propuestos de programación científica

1. Realizar un programa que calcule y muestre por pantalla los números primos menores que N, donde N será pedido por el programa e introducido por el usuario.
2. Realizar un programa que calcule la integral numérica aproximada de la función $f(x)=1-e^{-x}$ en el intervalo [0,L] donde L será pedido por el programa e introducido por el usuario. Utilizar el método de los rectángulos.
3. Realizar un programa que calcule la integral numérica aproximada de la función $f(x)=1-e^{-x}$ en el intervalo [0,L] donde L será pedido por el programa e introducido por el usuario. Utilizar el método de los trapecios.
4. Realizar un programa que pida el tiempo de duración de todas y cada una de las canciones de un CD en el formato *minutos:segundos* y realice y muestre el tiempo total de duración del CD en el mismo formato.
5. Realizar un programa que pida el D.N.I. del usuario y calcule la letra de control para mostrar el N.I.F.
6. Realizar un programa que pida el día, mes y año de una fecha y muestre el día de la semana (lunes, martes, ..., domingo) en que cae.
7. Realizar un programa que calcule la velocidad con que cae un cuerpo por un plano inclinado siendo los datos de entrada: la masa del cuerpo, la inclinación del plano inclinado y el coeficiente de rozamiento. El programa debe mostrar la velocidad a intervalos de 0,1 segundos y el tiempo de simulación lo decide el usuario.
8. Realizar un programa que calcule iterativamente el producto $\text{sen}(\pi/n)\cdot \text{sen}(2\pi/n)\cdot \text{sen}(3\pi/n)\cdot\ldots\cdot \text{sen}((n-1)\pi/n)$ siendo *n* un número natural mayor o igual que 2 a introducir por el usuario. El programa deberá comprobar también que la solución coincide con $n/(2^{n-1})$.
9. Realizar un programa que convierta un número N decimal introducido por el usuario a base octal. El número N será real.
10. Realizar un programa que dada una palabra introducida por el usuario, la deje centrada en la pantalla (horizontalmente).
11. Realizar un programa que pida los elementos de un vector de 10 elementos y, posteriormente, compruebe si existen valores repetidos, mostrando estos y sus posiciones dentro del vector.
12. Realizar un programa que calcule la media ponderada de una serie de valores. El usuario deberá introducir los valores así como la ponderación de cada uno de ellos.
13. Realizar un programa que sume los números enteros desde N1 hasta N2, donde N1 y N2 los deberá pedir el programa.

14. Realizar un programa que calcule el número de monedas de 500, 200, 100, 50, 25, 10, 5, 2 y 1 pts en una cantidad de dinero que introduzca el usuario de forma tal que tengamos el mínimo número de monedas.
15. Realizar un programa que realice la suma de dos matrices introducidas por el usuario.
16. Realizar un programa que calcule la traspuesta de una matriz introducida por el usuario.
17. Realizar un programa que calcule el producto de dos matrices introducidas por el usuario.
18. Realizar un programa que nos devuelva el mayor de las componentes de un vector introducido por el usuario y su posición original dentro del vector.
19. Realizar un programa que lea 10 número reales y los escriba ordenados.
20. Realizar un programa que calcule el desarrollo $1/1^2+1/2^2+1/3^2+...+1/n^2+...$ despreciando todos los términos siguientes a aquel cuyo valor absoluto sea menor que 10^{-6}. El programa debe comprobar que dicha serie aproxima a $\pi^2/6$.
21. Realizar un programa que calcule la serie de Fibonacci. El programa debe pedir el número de términos deseados.
22. Realizar un programa que calcule el cociente de dos números complejos.
23. Realizar un programa que resuelva la ecuación $f(x)=x^2-43=0$ por el método de Newton-Raphson. El programa debe pedir el punto de inicio del algoritmo x_0, el número máximo de iteraciones *n* y la aproximación deseada ε.
24. Realizar un programa que resuelva la ecuación $f(x)=x^3-2x-5=0$ por el método de bisección sucesiva. El programa debe pedir los valores extremos iniciales, el número máximo de iteraciones y la aproximación deseada.
25. Realizar un programa que lea una matriz de 10 por 10 y escriba la fila que cumpla que la suma de los valores absolutos de sus elementos sea mayor.
26. Realizar un programa que calcule el número de días pasados desde la fecha de nacimiento del usuario.
27. Realizar un programa que obtenga los factores primos de un número entero y nos diga si dicho número es primo.
28. Realizar un programa que calcule la media de *n* números y la desviación estándar.
29. Realizar un programa que simule la tirada al aire de una moneda 10000 veces y calcule el número de veces que sale cara y cruz, así como la longitud de la secuencia más larga de resultados iguales (cara o cruz) seguidos. Utilizar la función rand() para dar un carácter aleatorio a la simulación.
30. Realizar un programa que simule el movimiento de un cuerpo que cuelga en vertical sujeto por un muelle a lo largo del tiempo. El programa debe pedir como variables la masa del cuerpo, la constante de amortiguamiento

del muelle, la constante viscosa del muelle, el tiempo de simulación y el intervalo temporal con el que se quieren tener los datos. El programa irá sacando una tabla a lo largo de la simulación con el tiempo, la posición y la velocidad ada instante.

31. Realizar un programa que simule el movimiento de un péndulo. El programa debe pedir como variables la masa, la longitud del hilo, la constante de rozamiento con el aire (se supone que actúa con una fuerza proporcional a la velocidad), el tiempo de simulación y el intervalo temporal con el que se quieren tener los datos. El programa irá sacando una tabla a lo largo de la simulación con el tiempo, la posición y la velocidad ada instante.

32. Realizar un programa que realice una interpolación numérica lineal. El programa pedirá *n* puntos de coordenadas (x_i , y_i) y la coordenada *x* de un punto P. Por interpolación lineal usando los puntos vecinos a P, el programa debe devolver la coordenada *y* de dicho punto.

33. Realizar un programa que calcule el desarrollo 1-1/3+1/5-1/7+1/9-... despreciando todos los términos siguientes a aquel cuyo valor absoluto sea menor que 10^{-6}. El programa debe comprobar que dicha serie aproxima a $\pi/4$.

34. Realizar un programa que devuelva el polinomio de Hermite de grado *n*, $H_n(x)$ sabiendo que $H_0(x)=1$, $H_1(x)=2x$ y que la fórmula de recurrencia es $H_{n+1}(x)=2xH_n(x)-2nH_{n-1}(x)$.

35. Realizar un programa que obtenga el valor de $\pi/2$ a través del producto infinito $2/1\cdot2/3\cdot4/3\cdot4/5\cdot6/5\cdot6/7\cdot...$ con una precisión ε introducida por el usuario.

36. Realizar un programa que convierta un número N decimal introducido por el usuario a base hexadecimal. El número N será entero.

37. Realizar un programa que convierta un número N hexadecimal introducido por el usuario a base decimal. El número N será entero.

38. Realizar un programa que calcule el coste de una llamada provincial sabiendo que la tarifa de la llamada mínima son 15 pts, que cada minuto adicional se cobra según sea tarifa normal (laborable de 17 a 20 horas, 13,44 pts), tarifa punta (laborable de 8 a 17 horas, 15 pts) o tarifa reducida (6,71 pts), que el IVA (16%) no está incluido y que el total resultante se redondea a los céntimos. La facturación es por segundos: tasa de conexión + ptas por minuto/60 x número de segundos.

39. Realizar un programa que calcule el coste de una llamada metropolitana sabiendo que la tarifa de la llamada mínima son 11,40 pts, que cada minuto adicional se cobra según sea tarifa normal (laborable de 17 a 22 horas, 4,52 pts), tarifa punta (laborable de 8 a 17 horas y sábado de 8 a 14 horas, 4,52 pts) o tarifa reducida (1,64 pts), que el IVA (16%) no está incluido y que el total resultante se redondea a los céntimos. La facturación es por segundos: tasa de conexión + ptas por minuto/60 x número de segundos que excedan de una franquicia de 160 segundos.

40. Realizar un programa que calcule los biorritmos dando la fecha de nacimiento y la fecha actual. La teoría de los biorritmos especula que toda persona está influenciada por 3 ciclos que en su nacimiento tienen estas ordenadas: 1,2, 1,7 y 1, respectivamente. Estas cifras corresponden al nivel físico, intelectual y emocional que tienen forma de seno con períodos de 23, 33 y 28 días. Las amplitudes son 7, 14 y 7. El biorritmo para un día dado es el valor de las ordenadas de cada una de las curvas y el biorritmo total la suma de las 3 ordenadas.

41. Realizar un programa que admita una palabra por parte de un usuario y juegue al "juego del ahorcado" con otro usuario.

42. Realizar un programa que dado un número "romano" lo convierta a nuestro sistema de numeración.

43. Realizar un programa que pida un número entero positivo y lo convierta en número "romano".

44. Realizar un programa que cree una base de datos en memoria con nombres de personas y sus números de teléfonos, todo ello introducido por el usuario. Posteriormente, debe imprimirnos en pantalla una tabla con los nombres y sus teléfonos por orden de números de teléfono de menor a mayor y con una separación de una línea en blanco entre números de distinta provincia.

45. Realizar un programa que pida al usuario una palabra y calcule la suma de los códigos ASCII de las letras que componen la palabra.

46. Realizar un programa que calcule la distancia a una estrella por el método de triangulación. El programa debe pedir la distancia entre los puntos de observación y el ángulo de cada uno de ellos y calcular la distancia a la estrella. El programa, una vez mostrado el resultado, deberá quedar a la espera para que el usuario decida si quiere realizar un nuevo cálculo.

47. Realizar un programa que se "invente" un código secreto de manera que a cada letra minúscula le corresponda otra letra minúscula aleatoria, teniendo en cuenta que esta correspondencia debe ser una a una. Posteriormente, el usuario introducirá una palabra y el programa la codificará con el esquema desarrollado. Se utilizará la función rand() para la generación del código "secreto".

48. Realizar un programa que "desordene" aleatoriamente un vector de longitud *n* que deberá introducir el usuario. Se utilizará la función rand() para dar carácter aleatorio al "desorden".

49. Realizar un programa que saque por pantalla los números entre 1 y 100 que no sean múltiplos de 5.

50. Realizar un programa que lea del teclado un número entero positivo. Si es par lo divide por 2, y si es impar, lo multiplica por 3 y a continuación suma 1. Repetir el proceso hasta que se obtenga 1. El programa presentará por pantalla el número de elementos de la sucesión numérica generada.

51. Realizar un programa que cree un vector de 15 elementos enteros introducidos por el usuario y calcule el número de valores que sean pares, impares y nulos.